AMSTRAD

CPC 464, 664 Y 6128

PROGRAMACION ESTRUCTURADA

STEPHEN RAVEN

Comience

Camine hasta la parada del autobus

Espere hasta que llegue su autobus

Permanezca en la parada

Llega un autobus

Ese autobus le lleva al trabajo

Si

Súbase al autobus

no

Otra ronda de acciones

Finalice



AMSTRAD
CPC 464, 664 y 6128
Programación Estructurada

Stephen Raven



Título de la obra original

STRUCTURED PROGRAMMING
ON THE AMSTRAD COMPUTERS
CPC 464, 664 and 6128

Publicado en Gran Bretaña en 1985
por Micro Press
Castle House, 27 London Road
Tunbridge Wells, Kent

Imprime: Signo Impresores, S. A.
Albasanz, 27. 28037 Madrid
Depósito Legal: M. 41560 - 1985
I. S. B. N.: 84 - 86381 - 09 - 6
Edición en Español

AMSTRAD CPC 464, 664 y 6128
PROGRAMACION ESTRUCTURADA

Editado por RA-MA
Chiquinquira, 28 (COCUY)
28033 MADRID
ESPAÑA

Consultas referentes al libro
RA-MA, Chiquinquira, 28. Teléfono: 764.50.95 - 28033 Madrid

Traducción y Composición: CONORG, S.A.

Contenido

Sección A

Introducción al Amstrad CPC 464, 664 y 6128

EL CPC 6128

Cuando este libro salía para la imprenta el Amstrad lanzaba el micro CPC 6128 doméstico. Sus ventajas sobre el 664 es que está equipado con el doble de memoria de usuario RAM, un teclado diferente (que a mi gusto no es tan bueno como el del 664) y una nueva versión del sistema operativo industrial CP/M que le permite funcionar con un surtido más amplio de programas de gestión.

La buena nueva es que los programas escritos en BASIC para el 464 y el 664 funcionarán sin alteración en el 6128. Eso significa que debieran 'currar' todos los programas basados en el disco 664 y el 99,5% de los programas en cinta 464. **Eso significa también que puedes aprovechar todas las pistas y trucos de programación de este libro para aprender a usar tu nuevo 6128.**

Aunque el 6128 se suministra con 128K de RAM, sólo 68K están disponibles para los programas en BASIC. Eso es porque el BASIC del 6128 se diseñó originalmente para el 464 y el 664 que sólo tienen disponibles 64K. Para ayudarte a solventar este problema, Amstrad ha incluido algunos comandos BASIC extra que pueden implantarse desde el disco. Con ellos se te permite usar el 'repuesto' de 64K de RAM bien sea para almacenar imágenes de pantalla adicionales o bien como sistema rápido de archivo. Por el momento, no puedes usar esos 64K de respuesto para contener programas en BASIC.

Todos estos comandos extra quedan implantados en el sistema haciendo que se ejecute el programa en código máquina 'Bankmanager' suministrado en tu disquete. Si quieres utilizar esa RAM de repuesto para almacenar en ella imágenes de pantalla, el Bankmanager (gestor de las bancadas de memoria) te proporciona dos comandos adicionales.

| SCREENCOPY y | SCREENSWAP que te permiten conservar hasta 4 imágenes de pantalla en la memoria adicional y luego **canjearlas** sobre la pantalla. Eso puede ser util para juegos y animación en los que se prepara la nueva pantalla en la 'retaguardia' y luego se pasa a 'vanguardia' cuando es necesario. Cada imagen de pantalla se identifica con un número de bloque del 1 al 5. Para que se proyecte en el monitor es necesario moverla al bloque 1.

|SCREENCOPY hace que se **copie** toda una imagen de pantalla desde un bloque origen hasta un bloque destino. El contenido previo del bloque destino se pierde siempre al escribir encima. Por ejemplo, si quieres guardar tu imagen para un uso posterior, teclearías | SCREENCOPY, 1, 4. Con ello se copia la imagen desde el bloque 1 (que es siempre la imagen corriente) hasta el bloque 4 (que es parte de la RAM de repuesto).

| SCREENSWAP, 1, 4 justamente **canjea**, o intercambia, el contenido de dos bloques, sin sobreescribir ninguno de ellos. Por ejemplo | SCREENSWAP, 1, 4 pondría la imagen corriente en el bloque 4 y el contenido del bloque 4 pasaría a ser el proyectado en pantalla. Un ulterior | SCREENSWAP haría que las imágenes retornaran a la situación primitiva.

Si decides que quieres usar el repuesto de 64K de RAM para almacenar datos en lugar de imágenes en pantalla el 'gestor de bancada' te proporciona 4 comandos BASIC adicionales:

| BANKOPEN permite que se **abra** la bancada especificando cuantos caracteres habrá en cada registro. La máxima longitud es de 255 caracteres. | BANKWRITE hace que se **escriba** un registro, | BANKREAD te permite la **lectura** de un registro y | BANKFIND hace que se **halle** un registro que concuerde con un literal dado y devuelve el número identificativo del registro de manera que puedas efectuar una lectura posterior para conocer los datos registrados.

Todos los comandos aparte del de apertura de bancada necesitan especificar una variable entera para que en ella se contenga el 'código de estado' y una variable literal para reseñar los datos cuya lectura o escritura se va efectuar.

Capítuno Uno

La Máquina: su concepto y 'nutrición'

La computadora es una máquina muy compleja capaz de responder de una manera precisa y exacta a una serie de instrucciones.

Una **serie de instrucciones** es colocada dentro de la computadora por medio de un **programa.** Un programa se define como una lista de instrucciones que obligará a la computadora a efectuar un **trabajo** claramente definido. Este libro concierne primordialmente a como crear programas para el Amstrad CPC 664 y el CPC 464, con un estilo y una estructura que asegurará que tus esperanzas y objetivos para tus proyectos de programación se verán satisfechos. Tales principios generales son aplicables a todos los ordenadores, pero el Amstrad CPC 664 y el CPC 464 tienen como máquinas varias ventajas cuando se trata de confeccionar programas bien estructurados con la meta de que sean útiles en el hogar.

Dichas ventajas son:

1) Ambas computadoras permiten al usuario comunicarse con eficacia según un dialecto BASIC -el lenguaje mediante el cual damos instrucciones a la computadora- que es fácilmente legible con un mínimo de códigos y fórmulas sin significado patente.

2) La documentación que acompaña a las computadoras es exhaustiva y comprensible en sus descripciones de los comandos disponibles al usuario para dar órdenes a la máquina. Usando estos comandos, conjuntamente con este libro, la capacidad de programar en lenguaje BASIC solamente estará limitada por la imaginación del programador.

3) El equipo suministrado con el CPC 464, i.e. la lectograbadora de cinta y el monitor, significan una compra económica, asegurando además que no se harán exigencias adicionales en el aparato televisor doméstico.

Hay desventajas discernibles al usar una pantalla de TV cuando un requiere examinar cuál es la receta para el pastel de Navidad de la abuela y algún otro de la familia está viendo en ese momento concreto su ópera favorita en la televisión.

4) El CPC 664 tiene una ventaja clara al disponer de almacenamiento en disquete incorporado en el propio teclado. Este ordenador significa no solo una compra económica, sino debido a la unidad **ductora** de disco un elemento que ahorra tiempo al cargar y guardar programas, en relación con una unidad lectograbadora de cinta.

Para poder dar instrucciones a una computadora debemos presentárselas de una forma muy precisa, de manera que la computadora pueda entenderlas y luego ejecutarlas. Uno de los propósitos del teclado es el de entregar esas instrucciones a los círcuitos de tecnología avanzada contenidos dentro de la carcasa del ordenador, en lugar de dar las instrucciones a otra persona y luego que esa persona las realice. La persona interpretaría las instrucciones e incluso supliría los pedazos de información que no hubiera oido o entendido bien. Al igual que con cualquier otro ordenador, el Amstrad CPC 664/464 no puede interpretar instrucciones que se salgan de su lenguaje formal. Las instrucciones tienen que inscribirse de una manera -lo que se conoce como **sintaxis**- inteligible y acorde con el lenguaje BASIC CPC 664/464 que obedece las reglas prescritas en la máquina.

Para convertirse en un usuario provechoso del CPC 664/464, y desarrollar programas de aplicación productivos, con un estilo y estructura de programa que garantice la implementación con éxito de tus ideas, hay dos habilidades a las que debes aspirar. La primera es ser capaz de comunicarte con el CPC 664/464 en **su lenguaje,** que es un poco parecido al nuestro (al de los ingleses) pero más lógico y claramente más preciso en la forma de presentarlo. Si le entregas una instrucción que el CPC 664/464 no comprende, responderá con un mensaje -de la colección conocida como **mensajes de error**- que te dará una pista sobre la razón por la que el CPC 664/464 no puede comprender la instrucción recibida y por tanto no puede cumplimentarla. Es el uso informativo de esos mensajes de error lo que constituye uno de los elementos esenciales de un buen confeccionador de programas para el CPC 664/464.

La segunda habilidad a la que tienes que aspirar es la destreza para desarrollar tus pensamientos de tal manera que la aplicación esté planificada al detalle más minucioso, estando diseñada toda la estructura en la forma de **párrafos** que cuando se ensamblen conjuntamente llevarán a cabo la tarea requerida de manera exacta y repetitiva. Durante esta etapa de planificación, son instrumentos esenciales la pluma y el papel y el uso de diagramas. Eso no quiere decir que no usemos el CPC 664/464 para experimentar las ideas que tengamos, pero siempre después de ensayar la idea volveremos a planificar todo el proyecto antes de teclear el programa acabado en el ordenador.

A medida que tecleas tus instrucciones sería altamente improbable que no cometieras algunos errores. El CPC 664/464 responderá con sus mensajes de error. Con el fin de aprender rápida y eficazmente, será necesario que te tragues tu orgullo y aceptes que has de restringirte a las limitaciones del lenguaje BASIC CPC 664/464. Usa el apéndice 8 con los mensajes de error en el manual de usuario del CPC 664; los usuarios del CPC 664 deberán consultar el capítulo 7 parte 6 de su manual. Luego corrige las equivocaciones cometidas en las instrucciones. Sólo entonces serás capaz de obligar al CPC 664/464 a que cumplimente tus deseos.

Una computadora es 'paciente', dado que esperará a que tú tomes la iniciativa; repetirá la misma respuesta durante horas si no haces ningún cambio en las instrucciones que tiene **aprendidas** en su memoria. Si tales instrucciones incluyen partes que la computadora no puede interpretar expondrá en pantalla un mensaje de error. La computadora no puede corregir milagrosamente lo que no entiende. Es esencial tener en cuenta que el ordenador no cambiará (o siendo más preciso no sabe cambiar) su dialecto de comandos lógicos y precisos a tu dialecto que es -aunque me repugne decirlo- típicamente humano, y contiene por tanto numerosos elementos abiertos a diversas interpretaciones según la persona que te escuche.

Una computadora responde a listas de instrucciones que se conocen como programas; están tipificadas siguiendo un orden racional, y cada línea de instrucciones comienza con un **número de línea,** que va subiendo en magnitud a lo largo del programa. Con el fin de adquirir nuevas destrezas necesitamos concentrarnos en la capacidad del CPC 664/464 para responder a una sencilla línea de instrucciones de forma **inmediata:** es lo que se conoce como un **comando** en modo directo.

Hay varias utilizaciones para esta facilidad que nos ofrece el ordenador, mientras realmente estamos confeccionando un programa; pueden también usarse como una aplicación incluso antes de que hayas producido un programa.

Prepara tu ordenador y enciéndelo, de manera que aparezca el mensaje de **salutación** del fabricante expuesto en la pantalla. Teclea ahora las letras **CLS** o **cls** y pulsa la tecla marcada **ENTER** (que es la de color azul para indicarle que ya **vale**). El ordenador obedecerá la orden recibida y hará que se **aclare** la pantalla (CLS es abreviatura de CLEAR SCREEN), desapareciendo el mensaje de salutación y apareciendo la palabra READY, para indicar que está **presto** a recibir nuevos comandos y un cuadradito amarillo en la parte superior izquierda de la pantalla, que es el que 'corre' por toda la pantalla y por eso se denomina **cursor de texto.** Este es un ejemplo de un comando directo muy sencillo pero que muestra como lo obedece inmediatamente el CPC 664/464. Observa que el ordenador te permite que le inscribas las instrucciones tanto en mayúsculas como en minúsculas, lo cual es una excelente faceta de estas máquinas.

A partir de ahora cuando queramos que pulses teclas específicas, las realzaremos usando mayúsculas y negritas, e.g. **ENTER** significa que pulses la tecla de **vale,** marcada **ENTER** en el teclado (para indicar que realmente es **'adentro'**).

Los beneficios al usar comandos en modo directo son:

1) El uso de los comandos en modo directo permite al usuario de la máquina conseguir familiaridad con las facilidades disponibles y cierta perspectiva sobre su potencial. (Ensaya algunos comandos BASIC directamente ahora. Consulta el manual de referencia BASIC de tu máquina, o el manual del usuario y recuerda tomar nota de los requisitos **sintácticos,** o adquiere el excelente **prontuario** del Amstrad. Ensaya ahora por ejemplo **PEN 4** para que ese sea el color de la tinta con que está la **pluma** cargada).

2) El comando en modo directo te permite comprender el funcionamiento logical de la máquina y su velocidad de actuación, por ejemplo, para hacer cálculos aritméticos.

3) A medida que se desarrollan las habilidades de programación del usuario, los comandos en modo directo permiten usar el CPC 664/464 como un block de papel borrador pergeñando ideas en él para ensayarlas antes de incluirlas en un programa.

4) El uso de los comandos en modo directo ilustrará la frustración sentida al no haber almacenado las instrucciones para el uso posterior prolongado, como sucedería si las hubieramos dado como **instrucciones** (precedidas de un **número de línea**) para incluirlas en un programa.

El abuelo primitivo de las computadoras domésticas es la calculadora electrónica. Como era de esperar con todo el avance tecnológico, ahora están disponibles más y más posibilidades. El micro hogareño no es ninguna excepción. En lugar de tener meramente una serie de cifras en cristal líquido como una calculadora con lo que sólo se pueden exponer números, hay varias facilidades para EXponer la salida de información y puede mostrar toda clase de números, caracteres, y otros símbolos en la pantalla de monitor, en la cinta de la lectograbadora de cassette y si está conectada, en una impresora para obtener una copia impresa de la información. Por estas razones, los comandos -por ejemplo de cual es el resultado de un cálculo- han de ser explícitos. La instrucción para efectuar el cálculo tiene que incluir no sólo los operandos sino también en que forma y sobre qué equipo -monitor, impresora, cassette- ha de EXponerse, dónde -la posición en la pantalla, etc.- ha de hacerse la EXposición, y cuándo -en qué momento y durante cuanto tiempo- ha de estar mostrándose el resultado.

Teclea ahora las instrucciones exactamente como aparecen en la figura 1.1 y recuerda que hasta que le digas que **vale** pulsando la tecla **ENTER**, el ordenador meramente 'escucha y repica' lo que estás pulsando. Solamente al pulsar la tecla **ENTER**, interpreta que ya 'vale' o que 'adentro', y pasa a interpretar y en su caso cumplimentar lo que le has mandado. Para pulsar el signo + debes mantener pulsada la tecla de **turno** para mayúsculas y minúsculas -marcada **SHIFT** y pulsar la tecla correspondiente.

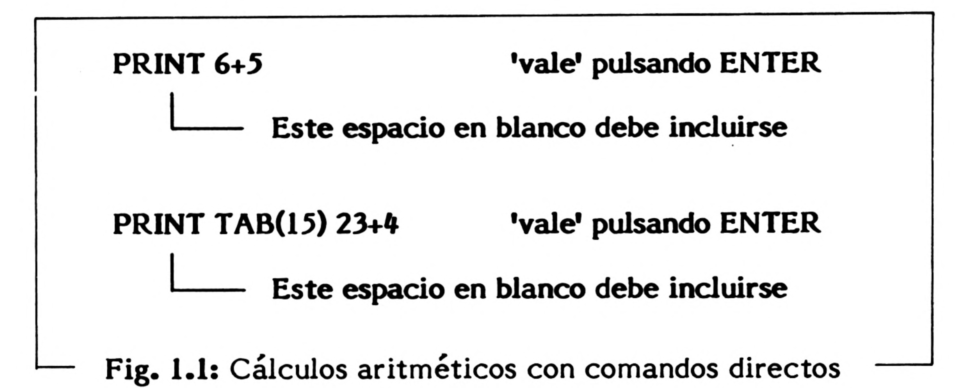


Fig. 1.1: Cálculos aritméticos con comandos directos

El resultado de las sumas aparecerá en la pantalla en el renglón inmediatamente inferior a la línea de comandos.

Experimenta ahora con números dentro de paréntesis. ¿Cuán larga es una línea? ¿Cuál es el número mayor que puedes colocar dentro de los paréntesis? Ensaya colocando una suma (dos números como sumandos) dentro de los paréntesis. No olvides volver a teclear el resto de la instrucción exactamente como en la instrucción original.

Probablemente has descubierto que el comando para **tabular** permite hasta 80 caracteres o dos filas horizontales sobre la pantalla. Se pueden usar fórmulas aritméticas adicionales para situar el resultado en la pantalla.

¿Qué clase de mensaje aparece en la pantalla si se omiten lo espacios en blanco? Si no has incluido el espacio en blanco -pulsando la llamada **barra espaciadora**- después de la palabra **PRINT** habrá aparecido en pantalla uno de dos mensajes de error. Con sólo la clave **PRINT** usada, el mensaje sería de **error sintáctico.** El ordenador no puede interpretar esa instrucción de acuerdo con las reglas del lenguaje BASIC. Si la clave **TAB** también se ha incluido, el mensaje de error dependerá del número que hayas colocado dentro de los paréntesis, que determina cual ha de ser la posición horizontal hasta la que debe saltar. Si el número es menor que 11 se expondrá el mensaje de error sintáctico. Si el número es 11 o mayor, se expondrá en pantalla el mensaje de **subíndice fuera de gama** que explicaremos con más detalle ulteriormente, pero efectivamente el CPC 664/464 está confuso en relación con lo que le mandas hacer.

Sin embargo, la mayoría de los errores sintácticos son el resultado de teclear erróneamente o de omitir un espacio en blanco que es esencial para la claridad de la instrucción, i.e. el espacio en blanco tiene el efecto de **delimitar** las palabras clave del BASIC CPC 664/464.

Los símbolos que el ordenador utiliza para hacer cálculos aritméticos no son los mismos con los que estás familiarizado. Son diferentes, únicamente por razones de legibilidad y por lo tanto serán más fácilmente reconocibles. Un resumen de los símbolos importantes y de los operadores aritméticos se muestra en la Fig. 1.2.

Símbolo	Operadores aritméticos y de relación
/	división
*	multiplicación
+	adición
-	sustración
\	división entera. El resultado será siempre un número entero (el cociente).
MOD	Da el residuo, es decir el resto de una división entera de dos números.
↑	Exponenciación i.e. elevar el número a una potencia, v.g: 4 ↑ 2 es cuatro elevado al cuadrado, o sea 16.
=	igual a
<	menor que
>	mayor que
< =	menor o igual a
> =	mayor o igual a
< >	distinto de

Figura 1.2: Los símbolos aritméticos y de relación

Todos los símbolos y combinaciones de la Fig. 1.2 aparecen en el teclado. Si el símbolo está en la parte inferior de la caperuza de la tecla puedes inscribirlo directamente. Si está en la parte superior tendrás que pulsar previamente y mantener pulsada la tecla de turno marcada SHIFT, mientras pulsas la tecla correspondiente al símbolo.

Experimenta haciendo cálculos adicionales de una índole más compleja. La computadora efectúa todas sus operaciones aritméticas siguiendo un específico orden para el cálculo: todas las multiplicaciones se efectúan primero seguidas de las divisiones, la suma y luego las restas. Como en la matemática tradicional, el uso de paréntesis está permitido y significa que esa parte del cálculo ha de efectuarse antes del resto de operaciones.

Expresión a mano	Expresión CPC 464
$\frac{(6 + 5)}{(6 \times (5 + 6))} = \frac{11}{6 \times 11} = \frac{1}{6}$	PRINT (6+5)/(6*(5+6)) 0.166666667

Figura 1.3: Comparación de expresiones aritméticas

La puntuación empleada para la instrucción de **EXposición de datos** de clave **PRINT**, es importante. Ensaya con esto:

```
PRINT 3+2,3-2,3*2
PRINT 3+2;3-2;3*2
```

Figura 1.4: Signos de puntuación al EXponer datos

Las **comas** (,) cuando se incluyen en un comando para exposición de datos permite que la salida de información sea proyectada en una de tres zonas a lo ancho de la pantalla. Dichas zonas se rellenan consecutivamente a partir del lado izquierdo; cuando las tres zonas están llenas el cursor automáticamente pasa a la siguiente línea.

El **punto-y-coma** (;) usado al exponer en pantalla datos numéricos hará que los números sean expuestos con un espacio blanco por delante y otro por detrás de cada uno.

Si sucede que inscribes una letra en lugar del número, verás que al decir que vale pulsando la tecla **ENTER** se expone un cero: has identificado una variable que tiene el valor de cero por ahora, lo que se explica en el capítulo 5 de Asignación de valores a variables. Por el momento, ensaya tecleando esto en el ordenador:

LET número=12:PRINT número

Un **dos-puntos** (:) indica al CPC 664/464 que vas a combinar una serie de comandos dentro de una misma línea de instrucciones. Una línea de instrucciones puede constar de hasta 255 caracteres, y no meramente un renglón longitudinal en la pantalla del monitor.

Vamos ahora a experimentar tecleando los siguientes comandos en tu computadora, y no olvides pulsar la tecla marcada **ENTER** para decir que ya 'vale', siempre que des por terminada una línea:

a) **MODE 0**

b) **MODE 1**

c) **MODE 2**

Pulsa y mantén pulsada la tecla de **turno** marcada **SHIFT** y la de **control** marcada **CTRL** mientras pulsas simultáneamente la tecla de **escape** marcada **ESC**.

d) **PEN 4:PAPER 2:BORDER 3**

e) **CLS**

Repite las instrucciones (d) y (e) usando números diferentes. Encuentra cualquier limitación que haya y produzca un mensaje de error. Ensaya cambiando el **modo** de gestión de la pantalla tecleando la línea (a) o la (c) una y otra vez con diferentes números como **parámetros**.

Para **restaurar** las condiciones iniciales en el CPC 664/464, (operación que corresponde al inglés 'reset') debes mantener pulsadas la tecla de **turno** marcada **SHIFT** y la de control marcada **CTRL** y pulsar al mismo tiempo la de **escape** marcada **ESC**.

Puedes incluir dentro de la misma línea de instrucciones varios comandos consecutivos mediante el uso del dos-puntos (:), lo que puede ser ventajoso tanto para razones de programación y también para ensayar ideas dando comandos en modo directo, de manera similar a usar un papel borrador.

FOR conta=1 TO 10:PRINT "CPC464":NEXT

Pudiera parecer que no hay límite a la extensión de una línea de instrucciones. Desafortunadamente no es así; debido a la construcción de la mayoría de los ordenadores hay un límite en el número de caracteres que pueden usarse en una sola línea. En el caso del CPC 664/464 ese límite es de 255 caracteres. Una desventaja a largo plazo con los comandos dados en el modo directo es la imposibilidad de almacenarlos en la memoria del ordenador para usarlos repetidamente. Consecuentemente tienen que volver a ser tecleados cada vez que se requiera esa misma acción.

Si los precedemos de un **número de línea** se convierten en lo que llamamos propiamente **instrucciones** y quedan 'prendidas' en la memoria del CPC 664/464 de manera que el usuario puede posteriormente repetirlas tantas veces como se desea. Una vez así almacenadas, tendremos lo que llamamos **programa** y para hacer que el ordenador lo siga nos basta mandarle que lo **rule**, o lo 'corra', usando el comando de clave **RUN**.

```
10 FOR conta=1 TO 10:PRINT "CPC464":NEXT
RUN
```

Aplicaciones con comandos en el modo directo

El uso más obvio para el CPC 664/464 dándole comandos en el modo directo es el equivalente al de una calculadora, con la capacidad para efectuar cálculos complicados simplemente pulsando una tecla. Sigue concienzudamente el ejemplo de la Fig. 15.

```
PRINT "Monto invertido = 25 KPts.":LET capital=25
PRINT "Tipo de interes = 12%":LET tipo=12
PRINT "Duracion en anos = 2":LET anos=2

LET intereses=(capital+tipo*anos)/100
LET balance=capital+intereses

PRINT "Los interses devengados son KPts = ";intereses
PRINT "El nuevo capital a loa 2 anos KPts = ";balance
```

Figura 1.5: Aplicación como calculadora mediante comandos

Esto demuestra un medio útil e informativo de efectuar una serie de cálculos. Examinando esa aplicación será posible que te percates del principio básico que el comando para que **EXponga** datos en realidad lo que hace es 'enviar' información, en este caso, a la pantalla. Es por tanto necesario instruir a la computadora con la misma información que estás observando en la pantalla; eso se efectúa diciendo que **haga** una variable igual a un valor, usando la palabra clave **LET**.

Vamos a considerar un enfoque diferente para usar comandos en el modo directo. **Restaura** las condiciones iniciales pulsando simultáneamente las teclas de **SHIFT, CTRL** y **ESC.**

Siguiendo las instrucciones de la Fig. 1.6 será posible que diseñes el plano de habitaciones, etc., usando el surtido de posibilidades gráficas disponible en el CPC 664/464.

```
MODE 0
WINDOW 1,20,4,4: PAPER 3: PEN 2: CLS
PRINT "1=ama 2=azul 0=borre"; : WINDOW 1,20,1,3
PEN 3: PAPER 1: CLS
ORIGIN 20,20,20,620,320,20
DRAW 600,0,2: DRAW 600,300: DRAW 0,300: DRAW 0,0
```

Figura 1.6: Dibujos con comandos en modo directo

Ahora experimenta por ti mismo. Si te metes en demasiada mezcolanza, **restaura** las condiciones iniciales y vuelve a comenzar desde el principio dando las instrucciones de la .Fig. 1.6.

Para que trace **rectas** usa cualquiera de las dos palabras clave **DRAW** (que significan trazar, dibujar). La de clave meramente **DRAW** usa **coordenadas absolutas** en las que la posición 0,0 es la esquina inferior izquierda de la pantalla. El área disponible para tus dibujos de planos de habitaciones, etc., tiene una coordenada horizontal x máxima de 600 y una coordenada vertical y máxima de 300 puntos.

Comando	Sintaxis
MOVE	MOVE coordenada x, coordenada y
MOVER	MOVER incremento x, incremento y el cursor se **mueve** con relación a la posición previa corriente
DRAW	DRAW coordenada x, coordenada y, color
DRAWR	DRAWR incremento x, incremento y, color el cursor **traza** a partir de la posición corriente previa

color = un número: 1=trazos en amarillo, etc

El color del trazo permanecerá hasta que sea subsecuentemente cambiado por otra instrucción.

Borra usando color=0 i.e. DRAWR 100,100,0 trazará una recta del mismo color que el **papel** o fondo, diagonalmente 100 unidades hacia arriba y 100 unidades hacia la derecha.

El rectángulo azul de la pantalla representa un **papel** para dibujar, una gradilla cuadriculada, que tiene una coordenada **x** máxima de 600 puntos y una coordenada **y** máxima de 300 puntos. La instrucciones de mover y trazar rectas se hacen colocando el cursor con relación a dicha gradilla. Las instrucciones de mover y trazar en coordenadas relativas son más útiles ya que no hacen referencia a esta gradilla sino que son cantidades de puntos a mover según el eje **x** y según el eje **y** con respecto a la posición que ocupaba anteriormente el cursor.

Figura 1.7: Uso de comandos en modo directo para dibujar

La instrucción para trazar **rectas** en modo relativo, de clave DRAWR usa coordenadas que están en relación con la última posición ocupada por el cursor de gráficos, i.e. para trazar una línea vertical ascendente a partir de la posición corriente, usas un **incremento de y** que equivalga a la longitud de la línea requerida mientras que usas un **incremento de x** cero. Para trazar una línea vertical descendente a partir de la posición corriente del cursor repites el proceso pero incluyendo un signo **menos** como un prefijo al valor del incremento y.

Para borrar una línea repite la serie previa de comandos que provocaron la línea o líneas no deseadas, habiendo primero movido el cursor al extremo inicial de la recta o rectas deseadas, y dando como parámetro de color en tu instrucción de trazado de rectas absolutas o relativas, el valor de **cero.**

Recuerda que si se te ocurre una idea, debes ensayarla por ti mismo. Experimenta a ver si puedes mejorar las aplicaciones que te proponemos. Una vez que hayas estudiado el resto de este libro vuelve a esa aplicación y analiza a ver si puedes convertir los **comandos** dados en modo directo en **instrucciones** que formen un programa para que luego posteriormente **guardarlo** en cinta cassette o en disquete, preparadas para el uso cuando y donde quieras.

Capítulo Dos

Gobierno del CPC 664/464 programándolo con instrucciones

Una aplicación informática consistirá de una serie de instrucciones aprendidas en la memoria del ordenador que llevan a cabo un trabajo o **'curro'** específico. Estas instrucciones puede por lo tanto que sean requeridas en numerosas ocasiones, haciendo totalmente impráctico teclearlas en el ordenador cada vez que se requiere. El almacenamiento de programas durante largo tiempo es una facilidad estremadamente valiosa y cuando se usa con el CPC 464 el mejor método es utilizar la lectograbadora de cinta situada en la parte derecha del teclado. El CPC 664 tiene una ventaja clara debido a la presencia de la una **unidad ductora de disco** incorporada en el sistema, que permite la gestión de **ficheros de información** de una forma más eficaz. El ejemplo del capítulo anterior no satisfaría estos requisitos y tendría que teclearse cada vez que fuere a utilizarse.

La aplicación que ahora vamos a considerar es de hecho una serie de tres programas que cuando se usen en combinación uno con otro pueden proporcionar un método efectivo y elegante de almacenar hasta un centenar de nombres y de números de teléfono en la forma de un **fichero de datos** informático. Una vez que esa información ha sido generada y archivada en un cassette, será posible agregar nombres y teléfonos adicionales en una etapa posterior, hasta el máximo permitido; consultar y revisar este listín telefónico personalizado con la mera pulsación de una tecla; buscar el teléfono dando el primer nombre o el apellido, y revisar todo lo reseñado en el listín correspondiente a un nombre concreto.

Antes de que realmente comencemos el trabajo de incluir este programa en tu ordenador CPC 664/464 debes tener en cuenta los siguientes asuntos:

1) El siguiente programa es un 'cimiento' para desarrollar y agregar a medida que tu capacidad de programación despliega todas sus alas. A través de cada capítulo se te harán sugerencias y se te presentarán ideas para que mejores aún más las facilidades disponibles al usuario del programa.

2) El diseño y la estructura construida en estos programas fue iniciada sobre papel antes de sentarse en el ordenador para teclearla. Si hubiéramos seguido otro enfoque en este tipo de libro, estarías leyendo sobre algo que se está confeccionando sin tener ni idea sobre el artículo acabado. Necesitas conocer primero el producto y ser eficaz en su utilización antes de que puedas determinar los cómos y los porqués de su creación y su aplicación.

Enciende tu computadora y asegúrate de que no hay nada en la memoria **restaurando** las condiciones iniciales usando las teclas de **SHIFT, CTRL** y **ESC.** Cuando inscribas una línea de instrucciones recuerda que has de decirle al ordenador que **vale,** pulsando la tecla **ENTER.** Antes de que comiences es posible mandar al ordenador que se coloque en el modo automático para que él vaya **numerando** las líneas de programa. El siguiente comando hará que comience a numerarlas a partir de la 10 y cada vez que digas 'vale' ese número de línea se incrementará también en 10:

AUTO número de línea, salto
e.g. AUTO 10,10

Para dejar el modo de numeración automático de línea basta pulsar la tecla de **ESC.** Para volver de nuevo a la numeración automática de líneas usa otra vez el comando de clave **AUTO** pero cambia el primero de sus **parámetros** para darle el número de línea que corresponde ahora en la continuación.

Parte primera: El menú de opciones

La meta de este programa es la de ser capaz de elegir entre generar/agregar datos al listín telefónico o usar las facilidades disponibles para consultar la información reflejada en él.

```
10 REM Listin Telefonico Personalizado
20 REM Steve Raven   Marzo 1985
30 REM Parte Primera Menu de Opcion
40 CLS: PEN 3
50 FOR tope=1 TO 22
60   PRINT TAB(tope) "Listin Telefonico"
70   NEXT
80 WINDOW 2,40,2,11
90 CLS: PRINT: PEN 2
```

```
100 PRINT " Que deseas hacer?"
110 PRINT: PRINT " <A> generar o agregar datos"
120 PRINT: PRINT " <B> utilizar el listin "
130 PRINT
140 PRINT "Pulsa la <letra> pertinente"
150 WINDOW 2,40,13,21
160 CLS: PRINT: PRINT
170 PRINT "Recuerda que debes haber creado "
180 PRINT
190 PRINT "el listin antes de utilizarlo."
200 PRINT
210 WHILE K$<>"A" AND K$<>"B"
220   K$=INKEY$
230   K$=UPPER$(K$)
240   PEN 1
250   WEND
260 IF K$="A" THEN RUN "Genere"
270 IF K$="B" THEN RUN "Utiliz"
280 END
```

Comprueba ahora cuidadosamente que has tecleado -unas cuantas líneas cada vez- mandando que **liste** las líneas de instrucciones ya registradas en la memoria, mediante por ejemplo:

LIST 10-200

Este comando hará que realmente **liste** todas las líneas que hayas tecleado (e indicado su final diciendo que 'vale') cuyo número de línea esté comprendido entre 10 y 200 ambos inclusive. Si falta una línea completa simplemente tienes que teclearla con su número y al decir que 'vale' el propio ordenador la incluirá en el sitio correspondiente, como podrás apreciar la siguiente vez que le mandes que **liste.** Si has cometido un error al teclear, por el momento lo mejor es que vuelvas a teclear otra vez toda la línea. Más adelante veremos métodos más efectivos de **revisar** y **modificar** -operación que en informática se denomina **editar** por aquello de 'sacar a la luz pública'- las líneas de programa, usando las facilidades con que está equipado el CPC 664/464. Una vez que hayas cotejado todas estas líneas pasa a comprobar las líneas desde la 210 a la 280 mandándole que las **liste.** Una vez que estés satisfecho con lo que has tecleado, te basta mandar que **rule** el programa, con el comando de clave **RUN** para ver la tarea que lleva a cabo.

Recuerda que no puedes realmente esperar todavía que el programa lleve a cabo su tarea de cargar uno de los dos programas que pide elija el usuario ya que todavía no los hemos confeccionado ni almacenado en la forma particular de almacenamiento que tienes en tu ordenador; así que usa este método para comprobar que no has cometido errores al teclear, i.e. que no aparece ningún mensaje de error en la pantalla al mandarle que lo **rule**, y así será si el programa ha sido inscrito exactamente como aparece en el libro.

El ordenador pasa a **ejecutar** las instrucciones del programa, obedeciéndolas, y por tanto pidiendo al usuario que elija una de las dos opciones. Encontrarás que sólo serás capaz de pulsar las teclas mencionadas o la tecla de **ESC** para que surtan algún efecto. Eso es lo que se conoce como 'validar' la respuesta del usuario a las preguntas que hace el programa, i.e. si ése no hubiera sido el caso, el programa no llevaría a cabo la tarea prescrita de generar o utilizar el listín telefónico. La razón de que hayamos mantenido todavía el uso de la tecla de **ESC**, es que de esa manera podemos regresar al modo normal y examinar el listado del programa. Si encuentras un mensaje de error manda que se ponga en **modo 1** y luego que **liste** y vuelve a cotejar la línea mencionada en el mensaje de error. Observa que los espacios en blanco son importantes y en caso necesario vuelve a teclear la línea de programa tal como aparece en el libro. Si no aparece ningún mensaje de error puedes hace que **guarde** el programa en su unidad de almacenamiento, cinta o disquete, mediante el comando de clave **SAVE**, que pasamos a explicarte. Para los usuarios del sistema CPC 464 deben seguirse las instrucciones enunciadas primeramente más abajo. Los usuarios del CPC 664 deben saltar la primera lista de instrucciones y pasar directamente a la segunda. Para detalles adicionales puedes consultar los respectivos manuales de usuario. Si después de volver hacer que **liste** el programa quieres volver a ver las instrucciones escritas en el color original, te basta mandar que adopte la **pluma 1**, mediante el comando **PEN 1**. Si no has cambiado el **modo** de gestión de pantalla antes de usar el comando para que **liste**, el listado del programa aparecerá en una de las 'ventanas' preparadas por el propio programa.

<u>Usuarios del CPC 464:</u>

1) Coloca un cassette en blanco dentro de la lectograbadora de cinta, repón a cero el contador de cassette, y bobina rápidamente el cassette de manera que el contador marque 0,25 aproximadamente.

2) Mándale que lo **guarde** mediante el comando SAVE "TELEFONO ", (y dile que **vale** pulsando la tecla marcada **ENTER**).

3) Sigue las instrucciones que aparecerán en la pantalla.

4) Verifica que el programa ha quedado correctamente grabado en el cassette, rebobinando la cinta de manera que el contador marque 0,23 por ejemplo, y mandándole que te muestre el **catálogo**, con el comando de clave **CAT**, apretando la tecla del cassette que corresponde al modo 'lectar' (marcada **PLAY**). A medida que va efectuando la 'lectura' aparecerá en pantalla un **bloque** de información examinada en el cassette, y aparecerá el mensaje:

TELEFONO block 1 $ OK

para indicar que está la grabación correcta (OK) y con ese nombre.

5) Si la acción que obtienes como respuesta al comando **CAT** no es la mencionada, vuelve a repetir el proceso para que **guarde** el programa en cassette.

Usuarios del CPC 664:

1) Coloca un disco nuevo o previamente **conformado** es decir estrenado o reestrenado, o como muchos dicen 'formateado' en la unidad **ductora** de disco (la popular 'disquetera'). Consulta el manual de la máquina sobre cómo efectuar el conformado de un disco mediante el comando **FORMAT**.

2) Manda que **guarde** el programa en disco dando el comando SAVE "TELEFONO ", y no olvides decir **vale** pulsando la tecla marcada **ENTER**.

3) Sigue las instrucciones que aparecerán en la pantalla.

4) Comprueba que el programa ha quedado grabado en el disquete mandando que te muestre el **catálogo**, con el comando de clave **CAT**. En la pantalla aparecerán todos los ficheros reseñados en el índice o 'directorio' de ese disquete.

5) Si en la lista no aparece el nombres TELEFONO , repite el proceso anterior hasta que la respuesta mostrada en el **catálogo** sea la mencionada.

Parte segunda: Generando y añadiendo datos al listín

Esta sección del programa para listín telefónico genera uno completamente nuevo o agrega nombres y teléfonos a un listín previamente creado.

Las líneas 10 a 190 sirven para la preparación del programa y como párrafo principal de control. Estas instrucciones pueden considerarse como el cimiento donde se construye el resto del programa, asegurándose que las **subrutinas** son citadas en el momento correcto y de acuerdo con la secuencia correcta.

```
10 REM Programa creador de listin telefonico
20 REM o agregador de nuevos nombres
30 REM y telefonos a un listin
40 REM previamente creado
50 REM Parte segunda   GENERE
60 :
70 LET t=100: LET contador=0: LET fichas=0
80 DIM nome$(t), apel$(t), tfno$(t)
90 LET nbr$="Nombre": LET apl$="Apellidos": LET tfn$="Telefono"
100 MODE 1
110 :
120 GOSUB 210 :REM @ Presentar opciones
130 IF k$="A" THEN GOSUB 310 :REM @ agregar datos
140 :
150 IF k$="C" THEN GOSUB 620 :REM @ crear listin nuevo
160 :
170 GOSUB 960 :REM @ guardar listin
180 GOSUB 1190 :REM @ explotar listin o PARE
190 END
```

Las líneas de programa de la 210 a la 300 están en relación con la **subrutina** que proporcionará al usuario del programa la información que él o ella necesita para elegir.

```
210 REM Presentador de opciones
220 LOCATE 1,2: PRINT "Elige el utensilio que requieras:"
230 LOCATE 5,4: PRINT "<C>reador de un Nuevo listin"
240 LOCATE 5,6: PRINT "<A>gregador de datos al listin"
250 LOCATE 1,8: PRINT "Pulsa <C> o <A>"
260 WHILE k$<>"A" AND k$<>"C"
270   LET k$=INKEY$
280   LET k$=UPPER$(k$)
290   WEND
300 RETURN
```

Las siguientes líneas de programa reflejan las instrucciones que facultan a la computadora la ejecución de las dos primeras posibilidades ofrecidas al usuario del programa en esta parte de la aplicación. La instrucción que sirve para incluir **memorandum**, de clave **REM** (abreviatura de remark: comentario) no es ejecutable y simplemente se incluye en los listados para hacerlos más comprensibles.

```
310 REM Agregador de datos al listin
320 CLS
330 LOCATE 5,24: PEN 5
340 PRINT "Mete la cinta en la lecto-grabadora"
350 LOCATE 1,1
360 FOR retraso=1 TO 1500: NEXT
370 CLS
380 OPENIN "fonos"
390 WHILE EOF = 0
400   INPUT#9,nome$(contador)
410   INPUT#9,apel$(contador)
420   INPUT#9,tfno$(contador)
430   PRINT contador,
440   LET contador = contador + 1
450   WEND
460 CLOSEIN
470 LET fichas = contador
480 WINDOW 1,40,1,23: CLS
490 PRINT nbr$,apl$,tfn$
500 LET contador=0
510 WHILE  contador <> fichas
520   PRINT nome$(contador), apel$(contador), tfno$(contador)
530   LET contador = contador + 1
540   WEND
550 WINDOW 1,40,24,24
560 PRINT "Pulsa <S> para que SIGA"
570 WHILE K$ <> "S"
580   LET K$=INKEY$
590   LET K$=UPPER$(K$)
600   WEND
610 LET K$="C": RETURN
```

Los usuarios del CPC 664 deben incluir la siguiente línea:

340 PRINT "Mete el disquete en la unidad ductora"

Las líneas 620 a la 950 son las que enseñan al ordenador como realizar la segunda de las posibilidades ofrecidas en esta parte de la aplicación.

```
620 REM creador de nuevo listin
630 WINDOW 1,40,1,25: CLS
640 PRINT SPC(10): PRINT "INscribe los datos": PRINT
650 PRINT nbr$,apl$,tfn$
660 WHILE contador < 100
670   LET fichas = contador + 1
680   WINDOW 1,40,11,11
690   IF K$=CHR$(32) THEN PEN 2
700   IF K$<>CHR$(32) THEN PEN 5
710   LET K$=""
720   PRINT "Cantidad de fichas: ";fichas
730   WINDOW 1,40,5,9
740   PRINT SPACE$(80)
750   LOCATE 1,2
760   LINE INPUT; nome$(contador)
770   LOCATE 14,2
780   LINE INPUT; apel$(contador)
790   LOCATE 27,2
800   LINE INPUT; tfno$(contador)
810   WINDOW 1,40,17,25
820   PRINT "Pulsa <ENTER> si el dato INscrito VALE"
830   PRINT: PRINT "Si no es correcto pulsa <ESPACIADOR>"
840   PRINT: PRINT "Si quieres que vaya al final pulsa <F>"
850   WHILE K$<>CHR$(13) AND K$<>CHR$(70) AND  K$<>CHR$(32)
860     LET K$=INKEY$: LET K$=UPPER$(K$)
870     WEND
880   CLS
890   IF K$=CHR$(13) GOTO 930
900   IF K$=CHR$(70) GOTO 920
910   IF K$=CHR$(32) GOTO 680
920   LET contador = 99
930   LET contador = contador + 1
940   WEND
950 RETURN
```

Una vez que el usuario del programa ha añadido nombres y teléfonos adicionales, o ha generado un listín telefónico nuevo es patente que la información tiene que archivarse de alguna manera para poderla usar en la parte tercera de la aplicación. Las líneas de programa de la 960 a la 1180 efectúan esta función.

```
960 REM guardador del listin creado
970 REM en la lecto-grabadora de cinta
980 PEN 5
990 WINDOW 1,40,1,25: CLS
1000 WINDOW 1,40,11,11
1010 PRINT SPC(5); "Cantidad total de fichas: ";fichas
1020 WINDOW 11,30,20,24
1030 PEN 7: CLS
1040 PRINT SPC(1) "Guardando Listin"
1050 PRINT: PRINT " en la cassette"
1060 WINDOW 1,40,1,8
1070 PEN 1
1080 LOCATE 1,1
1090 OPENOUT "fonos"
1100 LET contador = 0
1110 WHILE contador<fichas
1120   PRINT#9,nome$(contador)
1130   PRINT#9,apel$(contador)
1140   PRINT#9,tfno$(contador)
1150   LET contador = contador + 1
1160   WEND
1170 CLOSEOUT
1180 RETURN
```

Los usuarios del CPC 664 deben incluir las siguientes líneas:

970 REM grabar sobre ductora de disco
1050 PRINT:PRINT "sobre la unidad ductora de disco"

Las líneas 1190 a 1300 simplemente permiten al usuario del programa que acabe la sesión o que pase a la parte tercera de la aplicación y la utilice posteriormente.

```
1190 REM Utilizador de listin o PARADOR de curro
1200 WINDOW 1,40,1,25: CLS
1210 PRINT: PRINT "Pulsa la <tecla> apropiada"
1220 LOCATE 5,5: PRINT "<Z>apear := abandonar 'curro'"
1230 LOCATE 5,7: PRINT "<U>tilizar el listin"
1240 WHILE K$<>"Z" AND K$<>"U"
1250   LET K$=INKEY$
1260   LET K$=UPPER$(K$)
1270   WEND
1280 IF K$="Z" THEN CLS
1290 IF K$="U" THEN CLS: RUN "UTILIZ"
1300 RETURN
```

Repite el proceso de comprobar las líneas de programa para detectar los errores cometidos mandando al ordenador que lo **liste** y por el momento volviendo a teclear las líneas erróneas.

Si quieres comprobar como se comporta el programa hasta el momento, manda al CPC 664/464 que lo **rule,** usando el comando **RUN.** Para continuar con la confección del programa debes usar siempre este método: Pulsar la tecla de **ESC** (para escapar del control del programa), cambiar al **MODO 1** y hacer que **liste** el programa aprendido en memoria.

Usuarios del CPC 464: Una vez que hayas tecleado el programa tal y como está en estas páginas, coloca otra cassette en blanco en la lectograbadora y manda **guarde** este programa bajo el nombre de "Genere" dando el comando **SAVE** "Genere" y comprueba que ese nombre ha quedado reflejado en el **catálogo.**

Usuarios del CPC 664: Una vez que hayas comprobado que el programa es igual al que aparece en el libro, coloca el disquete en su unidad ductora y manda que **guarde** el programa bajo el nombre "Genere", dando el comando **SAVE** "Genere" y una vez que haya acabado la operación comprueba que está reseñado en el **catálogo** del disquete.

Parte tercera: Utilización del listín telefónico

La primera sección del programa, como en la parte segunda, es la preparación de los datos y el **párrafo** principal de control del programa.

```
10 REM Utilizador Listin Telefonico
20 REM Tercera parte   UTILIZ
30 :
40 LET t=100
50 DIM nome$(t), apel$(t), tfno$(t)
60 LET contador=0
70 LET nbr$="Nombre": LET apl$="Apellido": LET tfn$="Telefono"
80 MODE 1: PEN 1
90 :
100 GOSUB 200 :REM @ traer del cassette el listin
110 WHILE k<>4
120   GOSUB 380 :REM @ preguntar operacion requerida
130   ON k GOSUB 550,710,950,1230
140   WEND
150 REM 1500 repasador del listin
160 REM 2000 buscador de datos
170 REM 3000 enmendador de datos
180 REM 4000 zapeador de 'curro'
190 END
```

Las líneas 200 a 370 presentan la serie de instrucciones que harán que el CPC 664/464 **traiga** (y 'cargue' en memoria) los datos previamente guardados en el cassette o en el disquete por el usuario del programa, i.e. los nombres y teléfonos que haya en el listín.

```
200 REM traedor del listin desde el cassette
210 CLS
220 LOCATE 10,10: PRINT "Explotando el Listin de Telefonos"
230 LOCATE 5,24
240 PRINT "Mete la cassette en la lecto-grabadora"
250 LOCATE 1,1
260 FOR retardo = 1 TO 2500 :NEXT
270 LOCATE 1,1: PEN 2
280 OPENIN "fonos"
290 WHILE EOF = 0
```

```
300    INPUT#9,nome$(contador),apel$(contador),tfno$(contador)
310    PRINT contador,
320    LET contador = contador + 1
330    WEND
340 CLOSEIN
350 LET fichas = contador
360 PEN 1
370 RETURN
```

Los usuarios del CPC 664 deben incluir:

240 PRINT "coloca el disquete en su unidad ductora"

Las líneas 380 a 540 presentan en la pantalla la información que el usuario del programa requerirá para utilizarlo.

```
380 REM preguntador de operaciones
390 WINDOW 1,40,1,25: CLS
400 PRINT TAB(8) "Facilidades Disponibles"
410 LOCATE 5,3
420 PRINT "Mandame cada accion pulsando su numero"
430 PRINT "y dime que ya VALE pulsando <ENTER>"
440 LOCATE 8,7
450 PRINT "<1> Repase el listin"
460 LOCATE 8,9
470 PRINT "<2> Busque un telefono"
480 LOCATE 8,11
490 PRINT "<3> Enmiende una ficha"
500 LOCATE 8,13
510 PRINT "<4> Termine este 'curro'"
520 LOCATE 8,15
530 INPUT k
540 RETURN
```

Las **subrutinas** siguientes llevan a cabo las **tareas** elegidas por el usuario del programa. La función específica que ejercen queda indicada por la instrucción de **memorandum**, de clave **REM** que aparece en la primera línea de instrucciones de cada **párrafo.**

```
550 REM repasador del listin
560 CLS
570 PRINT nbr$,apl$,tfn$
580 WINDOW 1,40,3,18
```

```
590 LET contor = 0
600 WHILE contor < fichas
610   PRINT nome$(contor),apel$(contor),tfno$(contor)
620   LET pagina = contor MOD 14
630   IF pagina=0 AND contor>0 THEN GOSUB 1280 :REM @ ver si SIGA
640   IF pagina=0 AND contor>0 THEN WINDOW 1,40,3,18: CLS
650   LET contor = contor + 1
660   WEND
670 WINDOW 8,32,20,23
680 PRINT "Listin repasado del todo"
690 GOSUB 1280 :REM @ @ ver si SIGA
700 RETURN
710 REM buscador de telefonos
730 CLS
740 PRINT "Debo buscar por ... "
750 PRINT: PRINT "<A> Nombre "
760 PRINT: PRINT "<B> Apellido "
770 WHILE k$<>"A" AND k$<>"B"
780   LET k$=INKEY$
790   LET k$=UPPER$(k$)
800   WEND
810 PRINT
815 LET nb$="???": LET ap$="???": LET tf=0
820 IF k$="A" THEN INPUT "INscribe el nombre";nb$
830 IF k$="B" THEN INPUT "INscribe el apellido";ap$
840 PRINT: PRINT "    "; nbr$,apl$,tfn$
850 WINDOW 1,40,10,21
860 LET contor = 0
870 WHILE contor<fichas
880   IF k$="B" THEN GOTO 900
890   IF nb$=nome$(contor) THEN LET tf=1:
         PRINT contor;nome$(contor),apel$(contor),tfno$(contor)
900   IF ap$=apel$(contor) THEN LET tf=1:
         PRINT contor;nome$(contor),apel$(contor),tfno$(contor)
910   LET contor = contor + 1
920   WEND
925 IF tf<>1 THEN LOCATE 10,8: PRINT "No me has INscrito esa
    persona "
930 GOSUB 1280 :REM @ ver si SIGA
940 RETURN
950 REM Enmendador de datos telefono
960 CLS
970 PRINT "Dime la ficha para enmienda por su ..."
980 GOSUB 750 :REM @ pedir nombre o apellido y buscarla
985 IF tf<>1 THEN RETURN
990 WINDOW 1,40,22,24
1000 INPUT "INscribe 'numero ficha' a corregir";nid
1010 CLS: PRINT "Por favor reescribe la ficha entera."
1020 WINDOW 1,40,10,21
1030 PRINT nid; nome$(nid),apel$(nid),tfno$(nid)
1040 LOCATE 2,4: INPUT;nome$(nid)
1050 LOCATE 12,4: INPUT;apel$(nid)
1060 LOCATE 25,4: INPUT;tfno$(nid)
1070 WINDOW 1,40,1,25
1080 CLS
```

```
1090 PEN 7
1100 PRINT "Guardo el Listin en la lecto-grabadora"
1110 LOCATE 1,22
1120 OPENOUT "fonos"
1130 LET contador = 0
1140 WHILE contador < fichas
1150   PRINT#9,nome$(contador)
1160   PRINT#9,apel$(contador)
1170   PRINT#9,tfno$(contador)
1180   LET contador = contador + 1
1190   WEND
1200 CLOSEOUT
1210 PEN 1: PAPER 4
1220 RETURN
```

Los usuarios del CPC 664 deben incluir:

```
1100 PRINT 'Guardo el listin en la DUCTORA'
```

```
1230 REM zapeador de 'curro'
1240 CLS
1250 LOCATE 10,10
1260 PRINT "!Esto se ha terminado!"
1270 RETURN
1280 REM Indagador de si usuario manda que SIGA
1290 WINDOW 1,40,25,25
1300 PAPER 1: PEN 3
1310 CLS
1320 PRINT TAB(B) "Pulsa tecla para que SIGA currando"
1330 LET k$=INKEY$: IF k$="" THEN GOTO 1330
1340 PAPER 4: PEN 1: CLS
1350 WINDOW 1,40,1,25
1360 RETURN
```

Usuarios del CPC 464 únicamente: Una vez que estés satisfecho con el programa tal como lo has tecleado, manda que lo **guarde** en una tercera cassette con el nombre de "utiliz", dando el comando **SAVE** "utiliz" y comprobando que ha sido grabado correctamente diciendo que te muestre el **catálogo.**

Debieras tener a estas alturas tres cassettes con las tres partes de la aplicación archivadas por separado, un programa en cada cassette. Claramente, la aplicación es bastante aprovechable en esta forma, siempre y cuando el usuario esté preparado para colocar la cassette apropiada cuando el CPC 464 esté intentando **traer** (y 'cargar' en memoria) un programa concreto. Además de eso, el usuario no debe olvidar colocar una cassette en blanco dentro de la lectograbadora de cinta cuando quiera 'sacar' (poner fuera, EXponer) nombres y teléfonos en el cassette, o cuando quiera 'meter' (poner dentro, INponer) datos en el programa. Para mayor conveniencia, podría merecer la pena **traer** la parte 1 a la memoria del ordenador y luego **guardarla** en otra cassette en blanco, **traer** la parte 2 y **guardarla** en esa misma cassette a continuación, y hacer lo mismo con la parte 3.

El resultado sería una segunda copia de cada programa, pero archivadas en el mismo cassette, lo que evita el tener que cambiar las cassettes continuamente. Observa sin embargo que los nombres y los teléfonos deberán estar grabados en una cassette separada.

Si mandaras que te mostrara el **catálogo**, con el comando CAT, y con el cassette de programas dentro de la lectograbadora de cinta, la pantalla mostraría la información que se muestra en la Fig. 2.1. El símbolo **($)**, que indica siempre la **índole literal** de un dato como veremos, señala en este caso que la información almacenada en el cassette bajo la forma de un **fichero** con un determinado título, corresponde a un **programa** en BASIC, y no a un fichero de datos como sería el caso cuando examináramos el correspondiente a los nombres y números de teléfono que hay en el otro cassette, en cuyo momento el símbolo **(*)** nos señalaría que lo contenido en el fichero corresponde a **datos.**

```
Ready
CAT
Press PLAY then any key :

TELEFONO
Genere      block 1 $ ok
Genere      block 2 $ ok
Utiliz      block 1 $ ok
Utiliz      block 2 $ ok
```

Figura 2.1:

Catálogo del cassette de programas tal y como aparece en pantalla

Usuarios del CPC 664: Manda que **guarde** este programa en el mismo disquete usado previamente y de la misma manera, pero esta vez titula al fichero correspondiente con el nombre de "utiliz". Comprueba una vez más que ha quedado reseñado en el **catálogo** del disquete, donde deberán estar ahora grabados los tres programas confeccionados.

Restaura las condiciones iniciales de tu ordenador en la manera normal, pulsando las teclas de **turno**, de **control** y de **escape**, como ya te hemos indicado. Manda ahora directamente que **rule** el programa cuyo título es "TELEFONOS" para lo cual le darás el comando:

RUN "TELEFONOS"

Instrucciones operativas para el usuario

La primera vez que 'explotes' esta aplicación será necesario que generes un listín telefónico, para lo cual la secuencia de acciones será: manda con un solo comando que **traiga** y **rule** el programa de título "TELEFONOS" para lo cual basta dar el comando:

RUN "TELEFONOS"

y cuando se te ofrezca elige la opción indicada por la letra A. El programa automáticamente **traerá** y **rulará** la parte segunda de la aplicación. Una vez que se te dé la oportunidad elige la opción indicada por la letra C.

Usuarios del CPC 464: Unas palabras de aviso muy especiales: cuando uses uno u otro de los programas o trates los datos grabados en el cassette, asegúrate siempre que la cinta está rebobinada hasta la posición donde el programa o los datos comienzan, antes de intentar que el CPC 464 los **traiga** y 'cargue' en memoria.

Generación del listín telefónico

Al **inscribir** los nombres y los números de teléfono, teclea primero el nombre propio y díle que **vale** pulsando la tecla marcada **ENTER**, teclea luego el apellido y díle que **vale** pulsando la tecla marcada **ENTER**, y finalmente teclea el número de teléfono (y díle que **vale** pulsando la tecla marcada **ENTER**).

Una vez que el usuario ha generado su listín telefónico, le será posible mandar que **traiga** la parte 1ª de la aplicación y luego proceder directamente con la parte 3ª del listín telefónico para explotar las facilidades que ofrece de consultar los diversos nombres y teléfonos.

Sección B

La Familiaridad Incluye la Confianza

Capítulo Tres

Instrumentos BASIC de la Programación

Como resultado de haber pasado por el capítulo 2, es probable que ahora ya estés familiarizado con una cierta cantidad de palabras **clave** (i.e. los comandos y funciones) que constituyen el lenguaje BASIC, y que el programador usa para dar sus instrucciones al CPC 664/464. Este lenguaje se conoce como BASIC, o más precisamente como B.A.S.I.C., ya que es un **acrónimo** del inglés Beginners All-purpose Symbolic Instruction Code (y leyendo al revés veremos **Código Simbólico de Instrucciones de Propósito General y para Principiantes**). Como en todos los lenguajes hay en él una estructura muy definida: cada palabra no sólo encaja exactamente dentro de una cierta categoría, si no que se deletrea de una forma específica y se combina con las otras para formar un comando de acuerdo con unas reglas **sintácticas.**

Al caminar a lo largo de los tres programas que constituyen la aplicación listín telefónico, hemos hecho posible que pasemos a considerar las acciones o funciones que cada palabra implica y sus requisitos sintácticos.

La parte primera, el menú de opciones, es útil ahora por diversas razones. Las primeras tres líneas comienzan todas con la instrucción **memorandum**, de clave **REM** que es una abreviatura de la palabra inglesa 'remark': observación, comentario, nota; instruye al CPC 664/464 para que no tenga en cuenta el resto de esa línea, ya que es meramente una línea que sirve de recordatorio para que el lector del programa comprenda los diversos elementos del mismo, e.g. la línea 10 nos informa sobre el propósito de la propia aplicación, la línea 30 explica el propósito del primer programa de la serie de 3:

```
10 REM Listin Telefonico Personalizado
20 REM Steve Raven   Marzo 1985
30 REM Parte Primera Menu de Opcion
40 CLS: PEN 3
50 FOR tope=1 TO 22
60   PRINT TAB(tope) "Listin Telefonico"
70   NEXT
```

La línea 40 ilustra el uso de comandos que tienen un efecto inmediato, es decir **comandos** en el verdadero sentido de la palabra: mandatos u órdenes que se cumplimentan inmediatamente. Estas clases de palabras -son verbos de acción- son siempre muy útiles cuando se trabaja en el modo directo, como recordarás del capítulo 1.

El párrafo formado por las líneas de la 50 a la 70 es muy importante y muestra una de las capacidades más fundamentales de cualquier ordenador, i.e. la de ser capaz de **repetir** una tarea especifica, una cantidad 'preconfinada' de veces. Se señala mediante la instrucción **para...**, de clave **FOR**, que prepara una variable contadora (y véase el capítulo 5 sobre las variables) y en este caso el programador le ha dado el nombre 'tope'. El contenido de esta variable se fija inicialmente al valor 1, que es el valor de **comienzo.** La línea 60 manda que se **exponga** en la primera fila de la parte superior de la pantalla, y una columna hacia dentro contando desde la parte izquierda de la pantalla, un determinado mensaje. La línea 70 es la que marca donde termina el párrafo que ha de repetirse, al indicarle que proceda a examinar si debe dar o no **otra** ronda, mediante la clave **NEXT.** Por eso incrementará en una unidad el valor de la variable contadora y volverá a la línea 50 para examinar si el nuevo valor entra dentro de los límites establecidos en la instrucción **para....**

En consecuencia pasa ahora a exponer el mensaje en la nueva fila y columna señaladas. Como consecuencia de ser ahora la variable contadora 'tope'=2 empieza **tabulando** hasta esa posición (es decir colocando el cursor en la columna 2) y a continuación exponiendo el mensaje. Todo este proceso se repite **para** los valores establecidos, por lo que sólo termina cuando la variable 'tope' es superior en una unidad al número establecido como valor de **finalización** en dicha instrucción, i.e. una vez que tope=22 se terminan las rondas -y se ha formado por tanto un **bucle**- y se pasa a cumplimentar la instrucción reflejada en la línea que venga inmediatamente detrás de la que señala **otra** ronda si procede, de clave **NEXT.**

Hay dos maneras importantes de ajustar un párrafo del programa que ha de ejecutarse **reiteradamente** con el fin con que cuente progresivamente dando **saltos** mayores de la unidad en cada ronda. La siguiente instrucción hará que se dé un salto, o paso (en inglés **STEP**) de 5, cada vez que se dé una ronda:

```
50 FOR tope=1 TO 22 STEP 5
```

La siguiente instrucción hace que este bucle **para... otra** de programa cuente regresivamente dando saltos de -1:

50 FOR tope=21 TO 1 STEP -1

Hay otra posibilidad opcional para que el programador vea de manera más fácil cual es la instrucción **otra** de clave **NEXT**, que marca la culminación de un bucle iniciado por una instrucción **para...** de clave **FOR**. En ese caso se refleja la variable contadora de la siguiente forma:

70 NEXT tope

El párrafo de programa que va desde la línea 80 hasta la 200 del menú de opciones delimita dos recintos en la pantalla, cambia el color de la exposición, y hace que se exponga información esencial para el usuario del programa, sugiriendo que se requiere una cierta respuesta del usuario o usuaria para INponerla como dato del programa. El párrafo de programa desde la línea 210 hasta la 250 valida la respuesta recibida, asegurando que el ordenador sólo responde a una de las dos opciones ofrecidas al usuario del programa.

El párrafo comprendido entre las líneas 210 y 250, de manera similar a un bucle preconfinado **para... otra,** asegura que también se repite **condicionadamente** una serie de instrucciones del programa. Es decir, **mientras** que se cumpla una determinada condición, se ejecutará reiteradamente un cierto párrafo del programa. Así, la línea:

210 WHILE K$ < > "A" AND K$ < > "B"

que se lee: **mientras** que la variable literal llamada K sea **distinta** de la constante literal A y también sea **distinta** de la constante literal B, el programa ha de cumplimentar el párrafo de instrucciones que viene a continuación y cuya culminación viene señalada por la primera instrucción que encuentre de clave **WEND** (abreviatura de WHILE END) que en este caso es la línea 250. La expresión **condicional** que ha de cumplirse para dar cada una de las sucesivas rondas de este bucle condicionado puede variar de acuerdo con las exigencias del programa.

La **función** de nombre clave INKEY$ que se nombra en la línea 220 indica al ordenador que **inquiera** cual es la **tecla** más recientemente pulsada, y ese valor se manda sea hecho igual al contenido de la variable K$. Con eso se recoge un único carácter literal correspondiente a la tecla pulsada, y permitimos así al usuario del programa que manifieste su elección entre las ofrecidas.

La **función** de nombre clave **UPPER$**, que se nombra en la línea 230, es una posibilidad muy especial que convierte a **mayúsculas** el dato literal sobre el que se aplica. En este caso no importa pues que el usuario use minúsculas o mayúsculas, ya que el valor asignado a la variable K$ será siempre automáticamente cambiado a mayúsculas (UPPER significa de 'caja alta', como ocurría con las mayúsculas en la tipografía antigua).

Las líneas 260 y 270 del menú de opciones ilustran una de las capacidades fundamentales de todos los ordenadores, que es el de ejecutar o no un comando **si** se cumple una determinada condición, i.e. el contenido de una variable dada es igual a una constante dada.

260 IF K$ = "A" THEN RUN "Genere"
270 IF K$ = "B" THEN RUN "Utiliz"

La línea 260 dice que **'si'** la variable literal K es **igual** (tiene como valor o contenido) la constante literal A **entonces** -y únicamente en ese caso- ha de **rular** el programa que se menciona, para lo cual ha de traerlo previamente de la lectograbador de datos o de la unidad ductora de disco. La línea 260 es prácticamente igual, pero con una condición diferente y un título de programa diferente.

280 END

La última línea de esta primera parte es simplemente una manera conveniente de decirle que **fine** (finalice o acabe) la ejecución de un programa; en este caso simplemente está para una mayor claridad en la programación, dado que las líneas 260 y 270 aseguran una continuación del programa con la traida y ejecución de otro programa.

Vamos ahora a repasar los instrumentos del BASIC que hemos usado en las partes 2ª y 3ª de esta aplicación del listín telefónico para establecer los diversos comandos disponibles al confeccionador de programas. Todos ellos pueden ser encajados en las estructuras mostradas en la parte primera del programa, el menú de opciones, en las páginas previas de este capítulo. Estas estructuras que se usan como **comandos** cuando han de surtir efecto inmediato, reciben el término clasificativo más apropiado de **instrucciones** cuando meramente son aprendidas en la memoria de la máquina y luego sucesivamente ejecutadas una tras otra. La reiteración de una tarea varias veces ha sido mostrada por el uso de los bucles preconfinados **para... otra** y de los bucles condicionados **Mientras... FIn de Mientras**; el término típico para clasificar estos **párrafos** es el de secuencia repetitiva o reiterativa.

La tercera clasificación son las llamadas **decisiones**, o mejor condicionamiento de instrucciones que permiten incluir en un programa de ordenador comparaciones para constatar el valor de una variable o variables, y el cumplimiento de una u otra condición, i.e. y responder acordemente, tal y como se ha ilustrado mediante las instrucciones **Si... Entonces....** El término clasificativo más comúnmente usado es el de, **selección**, dado que el ordenador lleva acabo una acción especifica siempre que la condición seleccionada se vea satisfecha.

Probablemente la combinación más provechosa de palabras claves disponibles en el CPC 664/464, y que es la exigida para desarrollar programas bien estructurados profesionalmente, son las que permiten **desvíos** a subrutinas, que emplea las palabras clave **GOSUB**, para que **vaya-y-venga** al acabar esa tarea a un determinado párrafo del programa y la que indica que **vuelva**, de clave **RETURN**, que señala donde termina el párrafo que define la tarea a ejecutar. (Normalmente digo 'vayga' -para abreviar la de VAya Y venGA- a una línea concreta).

Normalmente se incluye una instrucción de **memorandum** en la línea señalada como **destino** del desvío efectuado, para explicar el objetivo de la subrutina. El párrafo de las líneas 120 a 190 muestra los elementos fundamentales de control de la parte segunda del programa de aplicación para el listín telefónico. Así en la línea 120 le decimos que **sí** K$="A" **entonces vaya -y venga** al acabar- a la línea 310, por lo que en realidad hacemos que se 'desvíe el curso' del programa hasta una determinada línea, y luego continúe a partir de esa línea progresivamente hasta que encuentre el comando donde le mandamos que **vuelva**, de clave **RETURN**.

```
120 GOSUB 210 :REM @ Presentar opciones
130 IF K$="A" THEN GOSUB 310 :REM @ agregar datos
140 :
150 IF K$="C" THEN GOSUB 620 :REM @ crear listin nuevo
160 :
170 GOSUB 960 :REM @ guardar listin
180 GOSUB 1190 :REM @ explotar listin o PARE
190 END
```

```
210 REM Presentador de opciones
220 LOCATE 1,2: PRINT "Elige el utensilio que requieras:"
230 LOCATE 5,4: PRINT "<C>reador de un Nuevo listin"
240 LOCATE 5,6: PRINT "<A>gregador de datos al listin"
250 LOCATE 1,8: PRINT "Pulsa <C> o <A>"
260 WHILE k$<>"A" AND k$<>"C"
270   LET k$=INKEY$
280   LET k$=UPPER$(k$)
290   WEND
300 RETURN
```

Al encontrarse con la instrucción para que **vuelva**, desviaremos de nuevo el curso seguido por el programa en ese momento y regresaremos a la línea directamente detrás del comando donde le mandamos que se desviara primitivamente, mediante la clave **GOSUB.** De esta manera la sección de programa comprendida entre las líneas 120 y 190 puede controlar cuál de las **tareas** ha de llevar a cabo de acuerdo con la selección por el usuario del programa. El segundo párrafo del programa, de las líneas 210 a la 300, ilustra una subrutina típica, que comienza con una instrucción de **memorandum**, de clave **REM** enunciando los objetivos de la subrutina con el propósito de hacer que la lectura del programa sea más fácil. La **tarea** se lleva a cabo cumplimentando la serie de instrucciones que aparecen en ese párrafo, cuyo final viene señalado por la instrucción que le manda que **vuelva**, de clave **RETURN.**

El mismo condicionamiento de las acciones que ves en las líneas 130 y 150, para que se efectúe el desvío a la subrutina **si** se cumple una determinada condición, también pudieras usarlo para condicionar la instrucción de **vuelva.**

Se pueden combinar las instrucciones para que **ubique** el cursor, de clave **LOCATE:** = sitúe, coloque, y la instrucción para que **EXponga,** de clave **PRINT:** = imprima, para producir algunas imágenes en pantalla interesantes y de estilo profesional.

Mencionaremos brevemente aquí -los detalles adicionales los presentaremos en el capítulo 6- que el CPC 664/464 tiene una característica especial por la cual le podemos mandar que **exponga** los datos a través de un determinado **cauce de salida,** que los lleve hacia un determinado **panel** o 'ventana' de la pantalla. Pueden elegirse uno cualquiera entre ocho y la sintaxis a utilizar es la de PRINT # **número identificativo,** y se suele decir que dicho número identifica un 'cursor de texto' dentro del recinto correspondiente de la pantalla. Inicialmente los ocho posibles (números del 0 al 7) están ubicados todos en la parte superior izquierda de la pantalla.

Si no se especifica ningún parámetro **# número,** decimos que se adopta el **prescrito para omisiones** que es el que corresponde al cursor cuyo número identificativo es el 0.

El comando para que **ubique** el cursor en un determinado sitio de la pantalla, de clave **LOCATE**, también puede ir seguido de un número identificativo del cauce de salida a emplear para situar el cursor; la sintaxis debe indicar entonces dos números separados por una coma: el primero de ellos representa el **número de columna** contado a partir del margen izquierdo de la pantalla; el segundo representa el **número de fila** contado a partir del margen superior de la pantalla y hacia abajo. En el ejemplo anterior se ha omitido el cauce de salida tanto para que **ubique** el cursor como para que **exponga** el mensaje, para mayor claridad; como se ha **omitido** el sistema tiene prescrito suponer que corresponde a 0. El máximo número de filas y de columnas depende del **modo** en que el ordenador gestione la pantalla en ese momento, y que puede ser el 0, 1 o el 2.

Combinando los comandos de **ubique** el cursor y de **exponga** datos a través de un **cauce** determinado, se puede conseguir imágenes en pantalla con aspecto realmente profesional.

La acción de **exponer** datos en pantalla o a través de un determinado cauce, mediante el verbo clave **PRINT**, puede verse modificada además usando el 'adverbio' de nombre clave **SPC** para indicar que **separe** con un cierto número de blancos o **espacios**, un dato del inmediatamente precedente. Actúa de manera muy similar al 'adverbio' de **tab**ulando, por lo que para distinguirlo muchos llaman **'seb**ular' a la operación de **se**parar con **bl**ancos, por contraposición a la de **ta**bular o saltar hasta un determinado tope de tabulación.

Otra **función** de índole completamente diferente pero que puede aprovecharse también para incluirla en un comando de exposición de datos para separar dos datos determinados mediante un cierto número de espacios en blanco, es la que utiliza la **función** de nombre clave **SPACE$**, **función** que al aplicarla a un determinado número entrega como resultado un valor constante literal formado por tantos blancos como indica dicho número. Por ejemplo, si mandamos que **haga** F$=SPACE$(5), cada vez que expongamos el dato variable F$ incluyéndolo en una instrucción de clave **PRINT** haremos que se expongan 5 espacios en blanco consecutivos.

El comando de clave **WINDOW:** = ventana, que en realidad corresponde a una **panelación** de la pantalla en recintos rectangulares, es una instrucción muy específica del Amstrad CPC 664/464. Un tratamiento muy cuidadoso de los cuatro parámetros que deben ir detrás de la palabra clave **WINDOW**, permitirá enmarcar una determinada área de la pantalla donde subsecuentemente hacer que se proyecten los datos a **exponer** mediante una clave **PRINT.** También es posible en el CPC 664/464 **identificar** hasta 8 paneles simultáneamente en la pantalla del CPC 664/464 mediante un número, que corresponde a un cauce de salida de información como ya hemos visto, y utilizar dicho número para que ubique el cursor y exponga los datos dentro del recinto delimitado. Eso de hecho significa que puedes generar 8 'ventanas' en la pantalla y usar un cursor diferente para exponer información en cada una de ellas. A lo largo de toda la parte primera de este programa de aplicación, y con el propósito de la simplicidad, hemos evitado el uso de **paneles** y de los cauces de salida asociados a ellos, manteniendo el 'prescrito para omisiones' que corresponde al número 0 y cuyo recinto está asociado a todo el área de pantalla. Para el comando de clave **WINDOW**, la sintaxis es pues:

WINDOW # cauce, izquierda, derecha, arriba, abajo

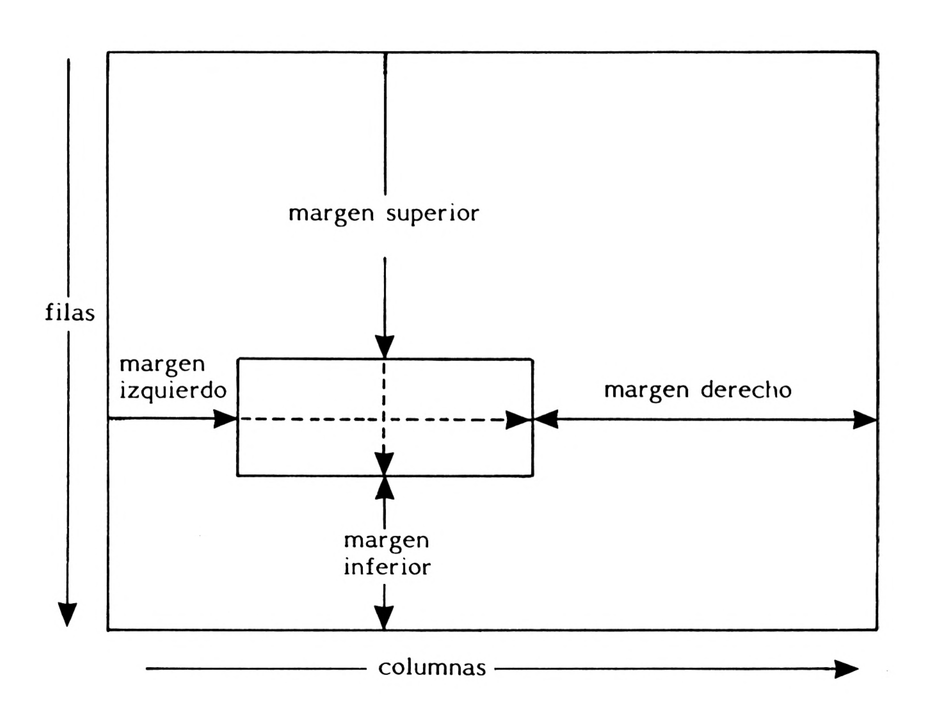


Figura 3.1: Situación de un panel en la pantalla

Para el aspirante a programador, el uso de código ASCII será cada vez más y más importante. Un **código** ASCII es un número que permite al ordenador cifrar o codificar la información que proviene del teclado o de otros canales de entrada al sistema. Las líneas de programa de Fig. la 3.2 utilizan este número establecido en la norma ASCII para la codificación, y así comprobar si se ha pulsado la **barra espaciadora.** En ellas, la **función** de nombre clave **CHR$(32)** entrega el **carácter** asociado al código 32 sobre el que se aplica la función, y que es el carácter literal generado cuando se pulsa la barra espaciadora. Ensaya por ejemplo mandando que se **exponga** el **carácter** que corresponde al **código** 69 en ASCII, tecleando PRINT CHR$(69). ¿Qué sucede?. Aparecerá expuesta en pantalla la letra E. De hecho la función **carácter** se aplica sobre un número cualquiera del 0 al 255, y entrega como resultado el carácter asociado a dicho **código** según la norma ASCII. El **código** para el espacio en blanco es el 32 como número decimal y como carácter es el 'espacio en blanco' que aparece en pantalla cuando mandamos que se exponga. Recuerda que el ordenador trata un espacio en blanco como cualquier otro carácter de su repertorio como la A, el punto, la admiración. Experimenta ahora comenzando con los siguientes ejemplos:

```
690 IF K$=CHR$(32) THEN PEN 2
700 IF K$<>CHR$(32) THEN PEN 5


5 MODE 0
10 PEN 6
20 FOR c=1 TO 800
30 LOCATE 10,10
40 PRINT CHR$(225)
45 LOCATE 10,10
50 PRINT CHR$(224)
80 NEXT
90 LOCATE 1,1
100 FOR c=1 TO 100
110 PRINT CHR$(227)+CHR$(217)+CHR$(218);
120 NEXT
```

Figura 3.2: El uso de códigos ASCII en un programa

Intenta escribir un breve párrafo de instrucciones, que incluya un bucle preconfinado **para... otra,** de manera que puedas observar los **caracteres** correspondientes a los códigos ASCII que van del 32 al 255. (Los que van del 0 al 31 no son caracteres 'visivos' sino que ejercen como caracteres de 'control').

Es necesario explicar brevemente al programador las pesadillas sin fin con las que se va a encontrar cuando utilice instrucciones en el programa para que el ordenador **vaya...** a un determinado número de línea del programa, usando la clave **GOTO,** cuando las emplee con poco cuidado. Puede hacer que el programa se vea atado a una serie de lazos y nudos, al hacerle que vaya de un sitio a otro sin ningún tino, por lo que su uso debe restringirse al mínimo. Sin embargo, aunque no es aconsejable si es frecuente encontrarlas como **salto** en el curso del programa que se efectúa de forma **condicionada** mediante una premisa **si... entonces...,** como en este ejemplo:

IF conta <100 THEN GOTO 90

Otros comandos y funciones con sus palabras claves y sintaxis correspondientes, serán tratados con mucha mayor profundidad en secciones adecuadas de este libro. Brevemente diremos que la familia de comandos para que **INponga,** o sea 'ponga dentro', del programa, determinados datos habitualmente según lo tecleado, utilizan como palabras clave **INPUT** y **INPUT LINE,** son las que permiten al programador introducir detalles e informaciones para ser tratados por el programa. En el caso del programa para listín telefónico, estos comandos se utilizarán antes de recoger la información que se haya previamente **EXpuesto,** o sea puesto fuera, sobre la cinta cassette o el disquete, haciendo que previamente se **abra** como fichero de salida uno de nombre determinado, que posteriormente mandaremos que se **cierre,** usando claves como **OPENOUT, PRINT #9** y **CLOSEOUT.** Los correspondientes comandos de apertura para la INposición de datos en el programa con claves **OPENIN, CLOSEIN, INPUT #9,** y la función que marca el **final de fichero** de clave **EOF,** utilizadas al recuperar datos previamente grabados son también un elemento esencial de cualquier programa de aplicación útil.

```
1120 OPENOUT "fonos"
1130 LET contador = 0
1140 WHILE contador < fichas
1150   PRINT#9,nome$(contador)
1160   PRINT#9,apel$(contador)
1170   PRINT#9,tfno$(contador)
1180   LET contador = contador + 1
1190   WEND
1200 CLOSEOUT
```

El párrafo comprendido entre las líneas 1120 y 1200 muestra la sintaxis de los comandos empleados para 'dejar' o depositar datos en una cinta de cassette o en un disquete, como archivo o fichero de datos:

```
280 OPENIN "fonos"
290 WHILE EOF = 0
300    INPUT#9,nome$(contador),apel$(contador),tfno$(contador)
310    PRINT contador,
320    LET contador = contador + 1
330    WEND
340 CLOSEIN
```

El párrafo de las líneas 280 a la 340 demuestra la sintaxis a usar en los comandos que permiten 'coger' o recuperar datos previamente almacenados en cinta o en disco e imponerlos como valores a las variables tratadas en el programa.

Depuración de Programas, Facilidades para Edición en el CPC 664/464, y cómo aprovecharlas

Usa la facilidad para traer y rular directamente un programa, para el menú de opciones tecleando el comando RUN "TELEFONO". Elige la opción A para que traiga del cassette y rule inmediatamente el programa llamado "Genere". Una vez cargado en memoria pulsa la tecla **ESC** que hasta que aparezca en pantalla el mensaje indicativo de que ha habido una inte**rrupción** en la línea 270. Haz que adopte el **modo 1** y que **liste** el programa aprendido en la memoria. El programa titulado como "Genere", i.e. la parte segunda de la aplicación de listín telefónico, y su longitud hará que la imagen vaya 'corriéndose hacia arriba' (desrrollando) por la pantalla. Pulsa la tecla marcada **ESC**, y el ordenador detendrá temporalmente el corrimiento de la imagen; pulsa cualquier otra tecla y reanudará el listado del programa. Usa sucesivamente dos veces la tecla de **ESC** y se detendrá por completo la proyección del listado del programa y aparecerá el mensaje de inte**rrupción** con la palabra inglesa ***BREAK***: = brecha, ruptura, proyectada en la pantalla. El CPC 664/464, estará ahora **escuchando** y **esperando** tus comandos o instrucciones desde el teclado. Experimenta con la sintaxis del comando para que **liste** el programa aprendido en su memoria, usando las siguientes alternativas:

LIST 100-150
LIST -100
LIST 150-

Los números que se dan como parámetros del comando **liste** en cada una de las expresiones se refieren a los números de línea del programa que en ese momento sea residente en la memoria del ordenador. Uno de los problemas que más frustración provoca al comenzar a programar tu ordenador es el problema constante de los mensajes de error que aparecen en la pantalla y que impiden llegar a una conclusión provechosa. También por otro lado solventar los errores del programa, o lo que se llama en la jerga técnica **depurar** un programa, (que se dice en inglés 'debugging': = quitar las pifias, o primitivamente los insectos) puede ser personalmente un proceso muy satisfactorio y ciertamente una manera muy provechosa de desarrollar una firme comprensión de los principios que subyacen en la programación de ordenadores.

La tarea ha de considerarse como un rompecabezas para cuya solución debe encontrarse un camino lógico de razonamiento. Debe decirse también que es en esta etapa donde la mayoría de las personas cejan en su tarea de aprender a programar. El único consejo que puedo dar es que tengas paciencia, estés preparado para experimentar con los diversos formatos y lo que es más importante pongas en práctica tus propias ideas de cómo puede resolverse el problema.

El orden lógico de operaciones al **depurar** el programa aprendido en la memoria del CPC 664/464 es el siguiente:

1) Mandar que **rule** el programa hasta el punto en que aparece el primer mensaje de error, anotar la clase de error y número de línea.

2) Mandar que adopte el **MODO 1**.

3) Mandar que **EDITE** esa línea (y editar en informática es hacer que la presente de manera especial para que puede **revisarse**).

4) Cotejar esa línea con la que aparece en el listado del libro para asegurarse que no se han cometido errores de tecleo. Si encuentras un error sitúa el cursor, con las flechas de cursor, al final del carácter erróneo y pulsa la tecla marcada **DEL**, para 'delectar' (suprimir, o tachar) el carácter o caracteres erróneos. Ahora vuelves a teclear el resto de la palabra o frase como debe ser.

5) Indica que has terminado la operación de edición de esa línea diciéndole que ya **vale**, para lo que has de pulsar como sabes la tecla marcada **ENTER**.

6) Manda ahora que **rule** el programa ya corregido y observa lo que sucede.

Si con eso no se resuelve ese error concreto, debes ahora examinar más profundamente la clase de error que es, por ejemplo, analizando el mensaje recibido que puede ser el de "**fin** de **mientras** inesperado" (Unexpected WEND) el cual te indica que ha encontrado una línea con esa clave WEND, sin que previamente hubiera encontrado la instrucción que le señalaba el principio de un bucle condicionado **mientras...**, de clave WHILE. Un mensaje que indique tal como "que falta **otra**", (NEXT missing) avisa que no encuentra la línea que señala la culminación de un bucle repetitivo para...

Estas clases de error requerirán que examines varias líneas consecutivas del programa por delante o por detrás de la línea donde el ordenador detectó el error.

Uno de los errores más comunes es el de 'error sintáctico', y habitualmente es el resultado de no haber dejado un espacio en blanco directamente después de una palabra clave.

El programa no responderá con un mensaje de error, pero no estará llevando a cabo lo que tu tenías previsto si, por ejemplo, una variable a la que denominaste 'tope' es en una línea posterior cambiada por un error de tecleado a digamos 'tupe': con eso habrás mencionado una variable denominada de esa manera, que realmente tendrá como contenido un cero, y con el efecto resultante sobre el programa. El término 'variable' se explicará en un capítulo ulterior.

Para resumir, los errores usuales al inscribir comandos e instrucciones en tu ordenador que se han tomado a partir del listado de un programa que aparece en un libro o en una revista, son los siguientes:

a) Diferencias en lo tecleado dentro de una línea específica de instrucciones, con respecto a lo que aparece en el listado.

b) No asegurarse que los imprescindibles espacios en blanco, preceden y siguen a cada una de las palabras clave del lenguaje utilizado en el ordenador.

c) Omitir el tecleo de alguna palabra clave.

d) No incluir una línea completa.

e) Olvidar pulsar la tecla indicativa de **vale**, marcada **ENTER**, al final de cada línea de instrucciones (y es un error muy frecuente).

f) No teclear los números de línea que indican al ordenador que se tratan de instrucciones y no de comandos, i.e. la línea habría sido así inscrita como línea de comandos en el modo directo y como consecuencia probablemente le hayamos obligado a ejecutarla de forma inmediata.

Mirando ahora hacia el futuro, cuando estés confeccionando tus propios programas, el problema de la **depuración** de los mismos tendrá un nuevo ángulo de perspectiva. Además de los errores sintácticos o de supresión de líneas, como resultado de tener los dedos agarrotados, también habrá errores en la 'lógica de razonamiento' de la estructura del programa y omisiones de circunstancias o datos, como resultado de ser novato en este juego (pero también ocurre aunque menos a los expertos). Siempre he visto que la mejor manera de quitarse de encima un mensaje de error es examinar cuidadosamente el listado del programa, pronunciando para si mismo los comandos que hemos dado, uno tras otro.

Métodos para alterar líneas de programa después de haber sido inscritas en la memoria

A continuación he enunciado los métodos más sencillos y más directos de alterar las líneas de instrucciones que ya están 'aprendidas' en la memoria del ordenador. Asegúrate antes que te has familiarizado con todas las modalidades del comando para que **liste** determinados trozos del programa.

1) Volver a teclear toda la línea de instrucciones otra vez, incluyendo el número de línea.

2) Aprovechar el comando para que **edite** una línea determinada, tecleando EDIT 'número de línea' y luego mover el cursor mediante las flechas izquierda y derecha hasta que estés situado encima del error cometido; a continuación usar una u otra de las teclas para borrar: la marcada **CLR** que es abreviatura de CLEAR:=anular, o la marcada **DEL**, que es abreviatura de DELETE:=delectar, suprimir, para eliminar el error. Luego teclear los caracteres correctos. El ordenador está colocado automáticamente en lo que se denomina la 'modalidad de **inserción**' de manera que automáticamente va desplazando el resto de la línea para poder 'intercalar' los nuevos caracteres o palabras clave que tecleas.

3) Mantén ahora pulsada una de las teclas de **turno,** marcada **SHIFT,** al mismo tiempo que pulsas la tecla para que suba el cursor, marcada con una flecha ascendente. El cursor que tiene forma de cuadradito amarillo hará aparecer un segundo cursor, que denominamos de **copia,** porque cuando pulsas la tecla marcada **COPY,** hará que el texto por el que está pasando este segundo cursor en cualquier parte de la pantalla que esté, sea copiado en la posición señalada por el primer cursor principal. De esta manera, será posible copiar aquellas partes de una línea de instrucciones que nos convenga y añadir instrucciones adicionales si lo necesitamos. No olvides que una vez que hayas completado una operación de copia, también debes indicarle que **vale,** pulsando la tecla marcada **ENTER,** para que dé la operación por terminada y vuelva a reunir los dos cursores en uno solo y lo sitúe por debajo de la última línea de instrucciones inscritas en la memoria.

Siempre será un ejercicio bueno y provechoso teclear programas en tu ordenador que copies de libros o revistas que hayas comprado; siempre y cuando prestes cuidado y atención e intentes saber lo que el programa está haciendo y como lo está haciendo. Para ello estudia los programas cuidadosàmente y sobre todo aprovecha las facilidades de edición para efectuar los cambios que se te ocurra y observar los efectos que obtienes al mandar que **rule** el programa que tu has adaptado. Recuerda obtener una copia del programa haciéndola que la **guarde** en cassette o en disquete antes de que comiences a hacer tus alteraciones, (que pueden ser desastrosas como es normal). Si sucede que tus cambios hacen que el programa sea irrecuperable y tengas que **restaurar** las condiciones iniciales, si has tenido la precaución de sacar la copia te basta con volver a mandar que la **traiga** del disquete o del cassette y la cargue en la memoria.

Sección C

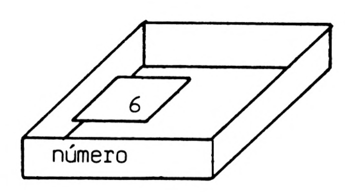
Los Principios del BASIC

Capítulo Cinco

Nombres de Variables o Rótulos para Lugares

Durante el proceso de examinar la posibilidad de emitir comandos en el modo directo y el listín telefónico como una aplicación, ha sido necesario mencionar de paso el concepto de usar **variables** para almacenar información vital para el propósito del usuario del ordenador en un momento concreto.

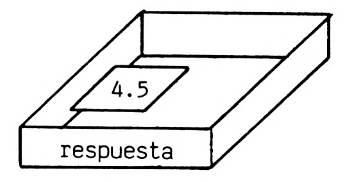
LET número=6



El valor es metido dentro del cajetín, en este caso el contenido será el número 6.

Se le da un nombre al cajetín, o lugar de la memoria, de manera que pueda directamente mencionar ese nombre para recuperar el valor contenido.

LET resul=(12+6)/4
i.e. resul=18/4
=4,5



El contenido del lugar de la memoria es el resultado de un breve cálculo.

El resultado de la suma puede ahora recuperarse pidiendo al ordenador que exponga el valor de la variable llamada 'resul'

Figura 5.1:

El valor de una variable representado como contenido de un cajetín, o lugar de la memoria

La manera más explicativa de hacer patentes estos **lugares de la memoria** donde temporalmente se almacena una información, que se denominan **variables**, es tratando de imaginar un cajetín dentro de la memoria del ordenador al que se le ha rotulado o dado un determinado nombre, de manera que su contenido puede examinarse, usarse o compararse con el contenido de otros cajetines, durante la ejecución de las instrucciones del programa, simplemente mencionando el nombre con el que se designa esa variable.

Para una demostración sencilla del uso de tales variables, ensaya los siguientes ejemplos de comandos:

```
LET a=12:PRINT a
LET número=34:PRINT número
```

Un nombre de variable puede constar de cualquier número de caracteres, letras o cifras, hasta un **máximo de cuarenta** siempre y cuando constituyan una retahíla continua con ningún espacio en blanco en ella. También debe **comenzar con una letra** y no con una cifra.

La instrucción de clave **LET** (usada para hacer verbos imperativos en inglés), que obliga a que el ordenador **haga** que el contenido de una variable tenga el valor que resulta de calcular una determinada expresión, aparece tantas veces en un lenguaje BASIC que la mayoría de los ordenadores no exige que se utilice explícitamente, i.e. es una palabra clave optativa que indica al ordenador que genere un cajetín o un lugar en su memoria, con un determinado nombre y con un determinado contenido:

nuro=56.67 es igual que
LET nuro=56.67

Si ahora le indicamos al CPC 664/464 que **exponga** el contenido del cajetín llamado **'nuro'** nos responderá mostrando en pantalla el número 56.67, que es el dato de la variable mencionada.

Algunos ejemplos adicionales de variables podemos verlos dando el comando de clave **LET**:

```
LET ENTER%=12.34:PRINT %
```

Observa que el ordenador responderá exponiendo el número 12 en la pantalla. El signo **porcentaje** (%) indica que la variable que lo lleva como **sufijo** es un cajetín especial en el que sólo pueden depositarse números **enteros**.

```
LET enter%=12.79:PRINT enter%
```

El CPC 664/464 responderá exponiendo en pantalla el número 13. Cuando se usa el símbolo **%** como sufijo de un nombre de variable, el ordenador automáticamente **redondeará** el valor que ha de depositar en ese cajetín de manera que corresponda al número entero más cercano al dado. Eso significa que si las cifras fraccionarias son menores de 0.5, el número quedará almacenado como el número entero inmediatamente inferior. Si las cifras fraccionarias son mayores de 0.5, el contenido del cajetín será aproximado al número entero más cercano que sea mayor que el dado.

```
LET nome$"YOLANDA":PRINT nome$
```

Una variable denominada con un nombre terminado en el símbolo **dólar** ($) corresponde a un lugar de la memoria, un cajetín, preparado especialmente para contener cualquier **sarta de caracteres** (en inglés se usa el término 'string': = retahíla, cordón, hilera, etc.) que esté delimitada entre una pareja de signos **comillas** (").

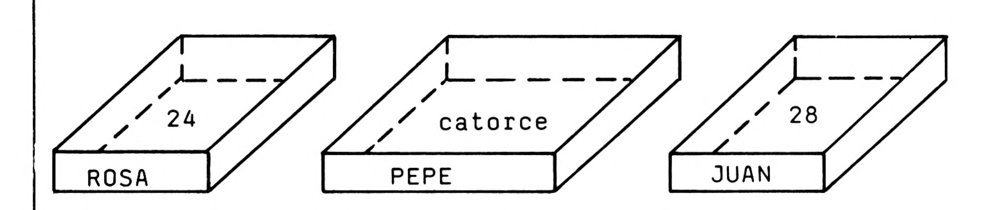
```
PRINT NOME$
```

Observarás que tienes el mismo valor expuesto que en el caso del comando anterior. Eso es debido a que el CPC 664/464 no distingue en el caso de nombres de variables entre letras mayúsculas y minúsculas. Así que ten en cuenta este hecho por dos razones:

1) No puedes denominar dos variables **diferentes** con el mismo nombre simplemente porque uno esté en mayúsculas y otro en minúsculas.

2) Es aconsejable que siempre uses mayúsculas o minúsculas para los nombres de variables a lo largo del programa.

La operación aritmética llevado a cabo en las líneas 60 y 70 (figura 5.2) requiere alguna explicación adicional. La operación designada por la clave **MOD**, que es abreviatura de MODule:= módulo, corresponde a lo que conocemos como 'congruencias', o sea el **residuo** o resto de dividir un número entero por otro. i.e. 24/3 es igual a 8 y no da ningún residuo. El signo de **operación** simbolizado por ,la raya invertida (\) corresponde al **cociente** entero entre dos números enteros, prescindiendo de las cifras fraccionarias que pudiera haber, i.e. 28 \3 da como cociente entero el 9, y queda como residuo 1. El resultado que obtendríamos es el 9.



Las instrucciones al ordenador se leerían:
Haga que en el cajetín denominado Rosa se deposite el número 24.
Haga que en el cajetín denominado Pepe$ se deposite la sarta de caracteres 'catorce', **literalmente** letra a letra.
Haga que en el cajetín denominado Juan se deposite el número 28.

La aritmética.	el resultado de:
24+28=52	Rosa+Juan=52
24*28=672	Rosa*Juan=672
28/28=1.16666667	Juan/Rosa= 1.16666667
24 MOD 3=0	Rosa MOD 3=0
28 \ 3=9	Juan \ 3=9
	Pepe$=catorce

El CPC 464/664 tiene que ser instruido de una manera muy precisa para que pueda efectuar las mismas operaciones aritméticas mencionadas anteriormente.

El programa	Lo expuesto en pantalla al rularse el programa
10 LET Rosa=24	
15 LET pepe$="catorce"	
20 LET Juan=28	
30 PRINT Rosa+Juan	52
40 PRINT Rosa*Juan	672
50 PRINT Juan/Rosa	1.16666667
60 PRINT Rosa MOD 3	0
70 PRINT Juan \3	9
80 PRINT pepe$	catorce
RUN	

Figura 5.2:

Demostración de variables como lugares para almacenar datos

La sarta de caracteres que es el valor dado a la variable literal pepe$ pertenece **constante** y al exponer su valor obtenemos **literalmente** 'catorce'. Ensaya el siguiente comando:

PRINT pepe$+pepe$

El resultado expuesto sería 'catorcecatorce'. Observa que no hay ningún espacio en blanco entre las dos sartas de caracteres porque el comando dice que se **empalme** (y se usa el término **concatenar**) esas dos retahílas de caracteres para formar una sola. El signo de la adición cuando se usa con operandos literales tiene pues una función diferente a cuando se usa con numerales.

Apreciarás cuando te conviertas en un programador eficiente que las variables son el núcleo principal de todos los programas. El uso de variables está ampliamente extendido porque con ellas se puede hacer un montón de tareas de forma extremadamente elegante. Vamos ahora a investigar la serie de programas para el listín telefónico y examinar como se utilizan las variables en él.

Parte primera: El menú de opciones

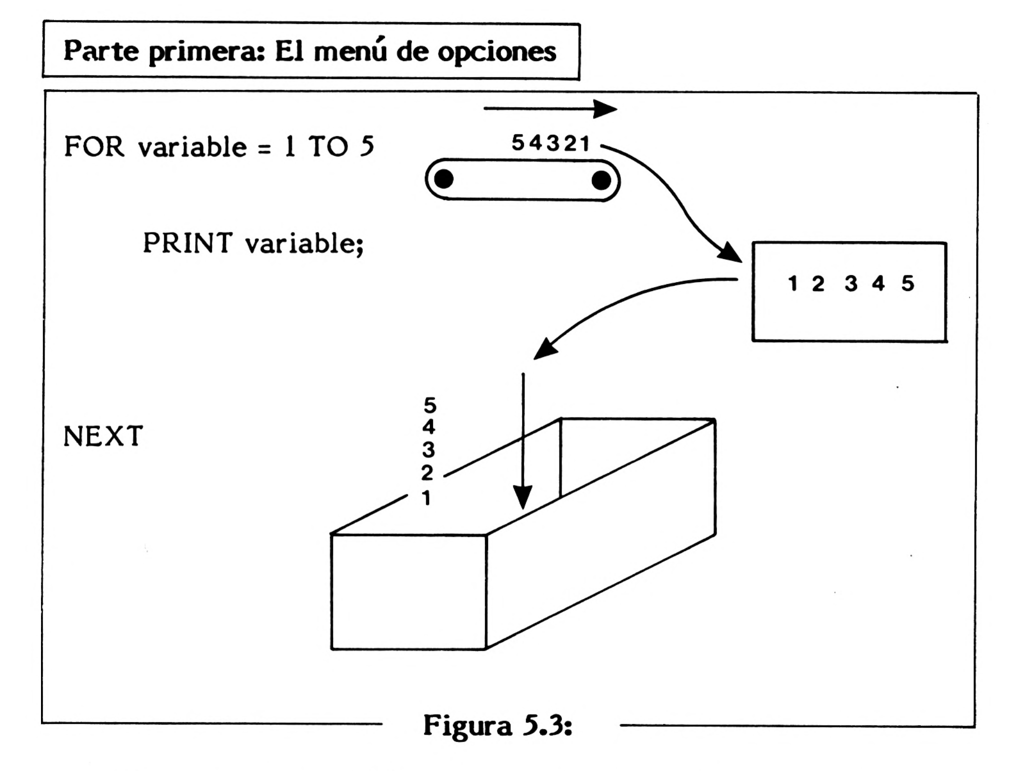


Figura 5.3:

El uso de una variable contadora en un bucle **para... otra**

La primera variable que nos encontramos es una muy especial dado que forma una parte intrínseca de todo bucle preconfinado **para... otra.** El contenido de la variable llamada 'tope' es repetidamente aumentado en una unidad entera hasta que dicho contenido alcanza el valor 22.

Las líneas 110 a 120 y 210 a 270 del programa para el menú de opciones ilustran otro uso de una variable:

```
100 PRINT " Que deseas hacer?"
110 PRINT: PRINT " <A> generar o agregar datos"
120 PRINT: PRINT " <B> utilizar el listin "

210 WHILE K$<>"A" AND K$<>"B"
220   K$=INKEY$
230   K$=UPPER$(K$)
240   PEN 1
250   WEND
260 IF K$="A" THEN RUN "Genere"
270 IF K$="B" THEN RUN "Utiliz"
```

La variable mencionada, de nombre K$, vincula la pregunta hecha al usuario del programa con los procesos comparativos condicionales que se efectúan dentro del mismo. Usando la función de clave **INKEY$** el contenido del lugar de la memoria rotulado como **K$** queda determinado por la tecla pulsada por el usuario. Si el usuario ha pulsado la letra A el programa 'genere' será traido del cassette o del disquete e inmediatamente ejecutado.

Parte segunda: Creación de un listín telefónico

Las siguientes tres líneas de programa demuestran un típico punto de arranque para la mayoría de los programas: la preparación de los valores iniciales de las variables.

```
70 LET t=100: LET contador=0: LET fichas=0
80 DIM nome$(t), apel$(t), tfno$(t)
90 LET nbr$="Nombre": LET apl$="Apellidos": LET tfn$="Telefono"

650 PRINT nbr$,apl$,tfn$
```

La línea 70 prepara la variable numeral de nombre t haciendo que el máximo número de nombre y números de teléfono sea su valor inicial. La variable de nombre **conta** servirá para contar el número de registros además de otras tareas similares a lo largo del programa y que exigen un valor inicial de 0. La variable **fichas** reflejará cual es el nombre y número telefónico que en cada momento está siendo tratado por el usuario del programa.

La línea 80 muestra una clase especial de variable **colectiva** que se conoce con los nombres de **ristra** o **tabla** (en inglés se usa 'array': = conjunto arreglado, ringla) y será comentado en detalle en el capítulo 14. De manera sencilla diremos que las ristras, o tablas monodimensionales, son un conjunto de variables **homogéneas** que están enlazadas de alguna manera y que poseen todas una manera única de identificarse. Por lo tanto el nombre genérico del colectivo será el mismo pero el **subíndice,** que es el número que se coloca dentro de los paréntesis, será único para cada elemento del conjunto. En este caso se **ocupa** espacio para **101** diferentes lugares de la memoria, de la misma clase y consecutivos, para depositar en ellos los valores correspondientes a las 101 variables. Observa que son: **nome$ (0), nome$ (1), nome$ (2)...** y así hasta **nome$ (100).** Estas variables podrían ser si se deseara,conjuntos de **dos dimensiones,** formando una retícula o cuadrícula, y así diríamos **nome$ (100,100)** generando una **ringla,** o tabla bidimensional, con elementos tales como **nome$ (0,1), nome$ (0,2), nome$ (0,3), nome$ (2,5)** hasta un total de 101 x 101 = 10201 elementos en ese conjunto.

La línea 90 del programa presenta una técnica muy aprovechable para generar variables literales cuyo contenido es una palabra, constante literal, que será usada una y otra vez a lo largo del programa. Siempre que se requiera dicha palabra se ahorra tiempo y esfuerzo simplemente indicando al ordenador que exponga el contenido de la variable correspondiente, tal y como se muestra en la línea 650 del programa.

Otra manera de utilizar variables es para establecer la condición que ha de cumplirse en los bucles condicionados **mientras que...** Esta forma de bucle cuyo comienzo se señala con la clave **WHILE** y su terminación con la clave WEND requiere un mecanismo para hacer que se cuente el número de rondas dadas ya sea progresiva o regresivamente; e.g. la línea 1150 de este ejemplo cuenta progresivamente dando rondas sucesivamente hasta que la condición establecida en la línea 1110 se satisfaga.

```
1100 LET contador = 0
1110 WHILE contador<fichas
1120    PRINT#9,nome$(contador)
1130    PRINT#9,apel$(contador)
1140    PRINT#9,tfno$(contador)
1150    LET contador = contador + 1
1160    WEND
```

Parte tercera: Uso del listín telefónico

El método final de colocar valores dentro de los lugares de la memoria rotulados con un nombre de variable es por medio de la instrucción para INponer datos a través del teclado de clave **INPUT**, como se muestra en las siguientes líneas.

```
1030 PRINT nid; nome$(nid),apel$(nid),tfno$(nid)
1040 LOCATE 2,4: INPUT;nome$(nid)
1050 LOCATE 12,4: INPUT;apel$(nid)
1060 LOCATE 25,4: INPUT;tfno$(nid)
```

Capítulo Seis

Ingreso de Datos en el CPC 664/464-Instrucción de Clave INPUT

Hasta ahora hemos explorado la capacidad del ordenador para almacenar la información en forma de datos **variables** creados dentro del propio programa, mencionando someramente la capacidad del usuario del programa o del operador para colocar un valor dentro del lugar de la memoria representado por una variable generada desde dentro del programa. Un ordenador sólo podría ser verdaderamente útil cuando pueda aceptar información procedente del operador o del programa de usuario, tratarla y proporcionar información elaborada a partir de la recibida, como resultado del proceso. La manera más simple de efectuar esto es que el ordenador prepare una o varias variables, lugares de la memoria, cuyo contenido estará determinado por la respuesta que el usuario del programa efectúe a una pregunta o a una serie de opciones ofrecidas en pantalla. La respuesta se convierte entonces en el contenido de ese lugar de la memoria, y decimos que se **IN**pone o se ingresa un dato como valor de esa variable. Un ejemplo sencillo pudiera ser:

```
10 INPUT "Dime tu nombre PorFa ";nome$
20 INPUT "Tu numero  suerte";nuro%
30 MODE 0:FOR conta=0 TO nuro%
40 PRINT nome$
50 LET tint=conta MOD 15
60 PEN tint: PAPER tint+1
70 NEXT conta
80 PRINT nome$+" !que tengas un buen dia!"
```

Este es un ejemplo con muy poco significado pero que ilustra unos cuantos hechos que concierne al uso preliminar de la instrucción para INposición de datos, de clave **INPUT:** = meter, poner dentro. Las reglas para las clases de variables y los sufijos usados en el nombre de las mismas, son las que ya vimos para cuando se les asignaba un valor directamente dentro del programa vía la instrucción de **haga...**, de clave **LET.**

La instrucción para INposición de datos, de clave **INPUT:** = poner dentro es muy similar a la instrucción para EXposición de datos (que es de clave **PRINT** pero debiera ser de clave **OUTPUT:** = poner fuera), y de hecho en una misma instrucción **INPUT** se puede poner un **mensaje** o aviso que aparecerá en la posición actual del cursor y que debe estar colocado después de la palabra clave **INPUT** y **entrecomillado** tal y como se muestra en las líneas 10 y 20 anteriores. La ventaja adicional es que esperará a imponer en la variable mencionada el dato tecleado por el usuario del programa.

El CPC 664/464 ha avisado al usuario del programa que requiere alguna información, prepara el tratamiento de acuerdo con las instrucciones programadas, que en este caso es un bucle preconfinado **para... otra**, con el que se repite la exposición del nombre y el cambio del color queda determinado por el número favorito del usuario del programa. Esta manera de utilizar la instrucción para INposición de datos, preparando la dirección y el formato de la EXposición de datos subsecuente, se utiliza ampliamente y se hace esencial en aquellos programas que van a ser usados por muchos y diversos usuarios. Por ejemplo, un programa puede diseñarse para aceptar números de teléfono, apellidos y nombres propios, pero otro usuario puede requerir sólo apellidos, direcciones y números de teléfono. Ambos son esencialmente la misma clase de programas, ambos tienen tres **campos** de información (es otra manera de llamar a las variables), pero la descripción y contenido de los campos variará. En esta situación es cuando una serie de instrucciones de clave **INPUT** son muy utiles para diseñar la forma de exponer en pantalla los datos correspondientes a esa aplicación. La información para el formato de la ficha y su distribución en pantalla queda luego almacenada en el fichero de datos en cinta, junto con los detalles personales y preparado para ser usado la siguiente vez que se requiera el programa de aplicación. Este tipo de programación puede verse como la generación de las instrucciones en un nivel conocido como los 'primeros principios' permitiendo al usuario del programa personalizar la aplicación programada de acuerdo con sus propios requisitos.

El esqueleto de un programa para tal propósito podría ser:

```
10 REM Informacion para aplicarla
110 LET contador=1
120 CLS:LOCATE 1,1
```

```
130 GOSUB 170
140 GOSUB 470
150 GOSUB 360
160 END
170 REM Generador informacion
180 PRINT "Titulo para este CARTEL de INgreso de  datos ";
181 INPUT "(maximo 30 caracteres.)";titulo$
190 IF LEN (titulo$)>30 GOTO 180
200 INPUT "Cuantos campos para datos";campo%
210 DIM detall$(campo%)
220 PRINT
230 PRINT "Cuando INgreses datos, NO USES ";: PRINT "comas."
240 PRINT
250 PRINT "Al inscribir los datos, cerciorate que ";
251 PRINT "las palabras del final de una linea no ";
252 PRINT "se adosan a las del principio de la otra"
260 PRINT
270 PRINT "Pulsa <ENTER> cuando hayas";: PRINT " terminado."
280 PRINT
290 WHILE contador<=campo%
300   PRINT "Numero del campo"; contador
310   INPUT "Teclea dato (max. 255)"; detall$(contador)
320   PRINT
330   LET contador = contador + 1
340   WEND
345 CLS
350 RETURN
360 REM Mostrar informacion en pantalla
380 PRINT TAB(5) titulo$
390 PRINT
400 LET contador=1
410 WHILE contador<=campo%
420   PRINT detall$(contador)
430   PRINT
435   IF VPOS(#0)>=17 THEN GOSUB 470
440   LET contador = contador + 1
450   WEND
460 RETURN
470 REM Rutina de pulsar cualquier tecla
480 LOCATE 1,24
490 PRINT "Pulsa cualquier tecla"
500 LET a$=INKEY$
510 IF a$="" THEN 500
515 CLS
520 RETURN
```

Como está en este momento, el programa permite al usuario diseñar una serie de frases y palabras que posteriormente serán mostradas en pantalla según una distribución estándar. Añade a este programa la posibilidad de guardar los parámetros que determinan las frases o sentencias sobre un fichero de datos almacenado en cassette o en disquete. La aplicación pudiera usarse para mantener de una manera organizada colecciones de información tales como recetas de cocina, operaciones graduales con aparatos mecánicos y tareas de mantenimiento, todas ellas pudiendo generarse y posteriormente archivarse en un disquete o cassette para ser consultadas siempre que se requieran.

Llegamos ahora al rasgo primordial y central de todos los buenos programas, es decir: la habilidad del programa para 'interactuar' con el usuario del programa. Identificando tres etapas en cualquier programa que sea interactivo con su usuario, el proceso de planificar y confeccionar el programa quedará simplificado hasta el extremo de convertirse meramente en una solución de problemas con un orden y razonamiento lógico. Las tres etapas que todos estos programas llevan intrínsecamente son:

1) El usuario del programa reacciona o responde a la información mostrada habitualmente en el pantalla del monitor.

2) El programa expone información en la pantalla para permitir que el usuario responda, habitualmente mediante la pulsación de una tecla.

3) El programa contiene las líneas de instrucciones que llevan a cabo las tareas correspondientes a las opciones ofrecidas al usuario mediante la pantalla.

Considera estos tres aspectos muy cuidadosamente. Los puntos 2 y 3 son claramente los que constituyen el propio programa, combinándose de manera que las metas establecidas para la aplicación pueden ser conseguidas por el usuario concreto del programa. Separando deliberadamente los elementos del programa se hace patente ahora que es posible escribir programas que estén claramente fuera de sintonía. Los detalles mostrados en pantalla pueden no llevar ninguna relación con lo que está sucediendo dentro del ordenador. Por ejemplo, en la pantalla pudiera verse 'pulsa la barra espaciadora' y nada sucede. Eso ocurriría si el programa sólo tuviera **salida de información**, i.e. instrucciones para exponer datos, y no contuviera las líneas del programa que aceptan la respuesta a la acción.

Tres sencillas líneas de instrucciones harán que se **aclare** la pantalla, se **ubique** el cursor, y se **exponga** el aviso en la pantalla:

```
10 CLS
20 LOCATE 10,10
30 PRINT "Pulsa la <BARRA ESPACIADORA>"
```

Desafortunadamente estas instrucciones no procesarán ninguna de las posibles reacciones del usuario del programa ya que simplemente no hemos reflejado en ellas la capacidad que tiene el ordenador para actuar así.

Con el fin de demostrar todavía más este enfoque pasemos a considerar dos ejemplos tomados del listín telefónico.

Las líneas 110 y 540 mostradas más abajo ilustran las líneas de programa relevantes. Ambos ejemplos siguen las mismas directrices: la exposición de comentarios para que el usuario del programa lea, junto con la sugerencia de lo que debe responder. Eso se logra eficazmente mediante instrucciones de EXposición de datos.

```
110 WHILE k<>4
120   GOSUB 380 :REM a preguntar operacion requerida
130   ON k GOSUB 550,710,950,1230
140   WEND

380 REM preguntador de operaciones
390 WINDOW 1,40,1,25: CLS
400 PRINT TAB(8) "Facilidades Disponibles"
410 LOCATE 5,3
420 PRINT "Mandame cada accion pulsando su numero"
430 PRINT "y dime que ya VALE pulsando <ENTER>"
440 LOCATE 8,7
450 PRINT "<1> Repase el listin"
460 LOCATE 8,9
470 PRINT "<2> Busque un telefono"
480 LOCATE 8,11
490 PRINT "<3> Enmiende una ficha"
500 LOCATE 8,13
510 PRINT "<4> Termine este 'curro'"
520 LOCATE 8,15
530 INPUT k
540 RETURN
```

Este ejemplo usa luego una instrucción de INposición de datos, de clave **INPUT**, para INponer como contenido de la variable K un número entre 1 y 4 ambos inclusive. En este caso la subrutina queda entonces completa y el control del programa **vuelve** a la línea 130 para que la respuesta obtenida sea tratada. Esta línea es una línea de instrucción especial que trabaja de manera parecida a la instrucción condicional **sí... entonces...** Si el contenido de K es 1 entonces el curso del programa se desvía hasta la subrutina que comienza en la línea 550. Si el contenido de K es 2 el curso del programa se desvía hasta la subrutina que comienza en la línea 710, y así sucesivamente. Con esto se han conseguido inmediatamente tres elementos fundamentales de un programa:

1) El usuario del programa puede interactuar con el programa.

2) El programa ha mostrado información en la pantalla de manera que el usuario pueda responder de forma correcta.

3) El programa es capaz de procesar la respuesta obtenida y la acción que se requiere se ve cumplimentada subsecuentemente al fijar la dirección a donde ha de desviarse el curso de la ejecución.

Es ahora pertinente lanzar un punto adicional que ha de considerarse. ¿Qué sucedería si el usuario del programa respondiera con una letra en lugar de con una cifra, o con una cifra menor de 1 o mayor que 4? Ensáyalo y observa lo que sucede.

Lo que debiera suceder es que si pulsas una letra y dices que **vale** pulsando **ENTER**, la respuesta del ordenador es un mensaje preguntándote si:

¿Vuelvo a hacerlo desde el principio? (?Redo from start)

Esta es una facilidad incorporada en el CPC 664/464 para que repita una instrucción de imposición de datos si en esa línea se especifica una variable numérica y el usuario teclea en su lugar datos literales, o sea sartas de caracteres. Si el número tecleado es menor que 1 o mayor que 4 toda la pantalla quedará en claro y todo el menú de opción volverá a ser expuesto. La razón radica en las líneas 120 y 130 que están colocadas dentro del bucle condicionado **mientras...**, es decir el párrafo entre las líneas 110 y 140.

En la línea 130, la instrucción que señala **ON... GOSUB...** hace sólo que se desvíe el curso del programa de acuerdo **con** el valor de la variable mencionada, siempre y cuando dicho valor corresponda ordinalmente a las líneas enumeradas después de la palabra clave **GOSUB.** De manera que si después, un usuario incorrectamente responde a la petición, el curso del programa pasa a la línea 140 y por lo tanto se vuelve a dar otra ronda a ese mismo bucle, i.e. se expone el menú de opciones esperando de nuevo una respuesta correcta por parte del usuario.

¿Qué sucedería si el usuario inscribe un número muy grande, digamos de varios centenares de millares? Desafortunadamente la respuesta del ordenador es un mensaje de error que inter**rumpe** el programa y que devuelve el control al usuario mostrando el signo de **presto** a recibir nuevas órdenes, con la palabra inglesa **READY**. Este es un problema que provoca frustración y el único remedio para él tal y como está el programa en la actualidad, es volver a mandar que **rule** el programa y volver a traer el fichero de datos de la cinta. Hay otro remedio y es el de añadir una línea de programa para indicarle que **con error** en la ejecución **vaya a...** mediante, por ejemplo:

35 ON ERROR GOTO 120

Recuerda que este programa así corregido deberás guardarlo en la cinta o en el disco según corresponda, porque en caso contrario la ventaja de esta línea adicional se perderá. Esta línea de programa indica al ordenador que revoque el efecto automático del ordenador cuando se encuentra un error de la ejecución, y deje por tanto de emitir el mensaje de error, ya que le manda que **con error vaya** a la línea 120, donde se vuelve a presentar el menú de opciones y se espera a la respuesta del usuario.

El procedimiento para hacer que tus programas estén 'a prueba de tontos' se conoce a menudo como la operación de '**validar** el programa para usuario'. Es una habilidad fundamental en la programación y merece la pena considerarla con profundidad.

El segundo ejemplo de exposición en pantalla y petición de respuesta al usuario con la capacidad de procesar dicha respuesta, sigue un procedimiento similar al mostrado en las líneas 110 a 540 con una diferencia importante en la manera en que el programa espera a que el usuario responda.

En lugar de usar una instrucción para **INponer** datos enlazada a un **desvío** de acuerdo **con** el valor IMpuesto, utiliza una combinación de un bucle condicionado **mientras que...** junto con la función de clave **INKEY$**, que **inquiere** cúal es la **tecla** pulsada, por lo que el lugar de realmente esperar a que todo el contenido de la variable así nombrada sea tecleado, meramente recoge el valor de la tecla últimamente pulsada y lo coloca como contenido de la variable.

```
80 WINDOW 2,40,2,11
90 CLS: PRINT: PEN 2
100 PRINT " Que deseas hacer?"
110 PRINT: PRINT " <A> generar o agregar datos"
120 PRINT: PRINT " <B> utilizar el listin "
130 PRINT
140 PRINT "Pulsa la <letra> pertinente"
150 WINDOW 2,40,13,21
160 CLS: PRINT: PRINT
170 PRINT "Recuerda que debes haber creado "
180 PRINT
190 PRINT "el listin antes de utilizarlo."
200 PRINT


210 WHILE K$<>"A" AND K$<>"B"
220   K$=INKEY$
230   K$=UPPER$(K$)
240   PEN 1
250   WEND
260 IF K$="A" THEN RUN "Genere"
270 IF K$="B" THEN RUN "Utiliz"
```

Al establecer en la línea 210 que ha de repetirse el bucle **mientras** que el contenido de la variable K$ sea distinto de A **y** sea distinto de B, el programa espera efectuando las acciones comprendidas en el bucle hasta que se cumpla esa condición. El problema de 'validación' se ha resuelto simplemente asegurando que ordenador continuará efectuando rondas del bucle mientras que no se alcance una de las dos respuestas requeridas. En respuestas más complicadas, la condición establecida para que se efectúe el bucle podrá ampliarse para abarcar más de dos opciones. Una vez que una de las respuestas correctas haya sido tecleada el programa procederá a efectuar la traida del programa de la cinta o del disco cuyo nombre o título sea el que ha determinado el confeccionador del programa para cada una de las opciones.

Para completar esta manera de enfocar la escritura de programas de aplicación, estudia las líneas de programa 730 a 940 y decide cuales son las series de línea que guardan relación con:

1) Instrucciones para exponer en pantalla.

2) Instrucciones para esperar la respuesta del usuario.

3) Instrucciones para tratar en el programa la respuesta procedente del usuario.

```
730 CLS
740 PRINT "Debo buscar por ... "
750 PRINT: PRINT "<A> Nombre "
760 PRINT: PRINT "<B> Apellido "
770 WHILE K$<>"A" AND K$<>"B"
780   LET K$=INKEY$
790   LET K$=UPPER$(K$)
800   WEND
810 PRINT
815 LET nb$="???": LET ap$="???": LET tf=0
820 IF K$="A" THEN INPUT "INscribe el nombre";nb$
830 IF K$="B" THEN INPUT "INscribe el apellido";ap$
840 PRINT: PRINT "   "; nbr$,apl$,tfn$
850 WINDOW 1,40,10,21
860 LET contor = 0
870 WHILE contor<fichas
880   IF K$="B" THEN GOTO 900
890   IF nb$=nome$(contor) THEN LET tf=1:
895   PRINT contor;nome$(contor),apel$(contor),tfno$(contor)
900   IF ap$=apel$(contor) THEN LET tf=1:
905   PRINT contor;nome$(contor),apel$(contor),tfno$(contor)
910   LET contor = contor + 1
920   WEND
925 IF tf<>1 THEN LOCATE 10,8: PRINT "No me has INscrito esa persona "
930 GOSUB 1280 :REM a ver si SIGA
940 RETURN
```

Ahora compara tu respuesta con la que te doy en la Fig. 6.1. Tengo esperanzas de que sea la misma o similar:

```
                                        Mostrando instrucciones por Pantalla
730 CLS
740 PRINT "Debo buscar por ... "
750 PRINT: PRINT "(A) Nombre "
760 PRINT: PRINT "(B) Apellido "         Mecano espera una respuesta

770 WHILE K$<>"A" AND K$<>"B"
780   LET K$=INKEY$
790   LET K$=UPPER$(K$)
800   WEND                               Mecano procesa la respuesta

810 PRINT
815 LET nb$="???": LET ap$="???": LET tf=0
820 IF K$="A" THEN INPUT "INscribe el nombre";nb$
830 IF K$="B" THEN INPUT "INscribe el apellido";ap$   Imagen en pantalla de la respuesta procesada

840 PRINT: PRINT "   "; nbr$,apl$,tfn$
850 WINDOW 1,40,10,21                    Procesando y mostrando el resultado del trabajo

860 LET contor = 0
870 WHILE contor<fichas
880   IF K$="B" THEN GOTO 900
890   IF nb$=nome$(contor) THEN LET tf=1:
      PRINT contor;nome$(contor),apel$(contor),tfno$(contor)
900   IF ap$=apel$(contor) THEN LET tf=1:
      PRINT contor;nome$(contor),apel$(contor),tfno$(contor)
910   LET contor = contor + 1
920   WEND                               Rutina: Pulsar Cualquier Tecla

930 GOSUB 1280 :REM a ver si SIGA
```

Figura 6.1:

Las subrutinas divididas en actividades específicas
Parte tres -subrutina de búsqueda y selección

Mientras estamos mirando esta subrutina concreta de la parte tercera del listín telefónico, puedes volver al programa original y hacer que el ordenador lo traiga a su memoria. Usa la tecla de **ESC** para producir una interrupción en la ejecución y manda que se **suprima** la línea 720, con el comando de clave **DELETE.** Ahora usa el programa varias veces respondiendo reiteradamente al menú para la opción de búsqueda y selección ¿Qué sucede una vez que se elimina la línea 720? Es un punto importante del proceso: una vez que la opción ha sido usada previamente en cada sesión particular, el programa funcionará pero no estará haciendo lo que se esperaba hiciera.

Sección D

Desde Pequeños Bloques hasta Programas Estructurados

Capítulo Siete

Planificación Estructurada de las Aplicaciones con el CPC 664/464

Veamos una vez más el listín telefónico personalizado como un modelo, para que nos guíe a través del proceso lógico de planificación de las aplicaciones de esa clase. Recuerda primero que al confeccionar y desarrollar tu propia aplicación, el trabajo con lápiz y papel de planificarla es la primera etapa. Este libro ha explicado el proceso de confeccionar un programa a partir de una situación desde el primer momento, de manera que ahora te es posible examinar el proceso de creación desde el punto de vista del creador. En otras palabras, cuando tengas tus propias ideas para aplicaciones del CPC 664/464 debes primero **imaginarte** exactamente lo que requieres, y luego después pasar a llevarlo a cabo. El escenario está ahora preparado, se ha examinado la aplicación como un todo, y se ha usado; debe **desglosarse** ahora en partes para poder **analizarla** con más detalle.

La idea debe surgir primero, y este es un proceso de pensamiento. En el caso que estamos examinando, la idea surgió de la necesidad de archivar un centenar de números telefónicos con la posibilidad de consultarlos o elegirlos por apellido o por nombre propio, y la posibilidad de eliminar ese rimero de notas de papel que hay alrededor del teléfono.

Una vez que la idea está formulada, ¿cuál será las **tareas** primordiales que llevará a cabo el operador al explotar el programa de aplicación?

1) La generación del listín telefónico inicial.

2) El agregamiento de nuevos nombres y números a un listín previamente generado.

3) La utilización del listín con el fin de consultarlo, buscar y elegir, y observar los nombres y números de teléfono.

4) La revisión (y en informática le llamamos **edición**) de nombres y números previamente reflejados en el listín.

Entre las tareas enunciadas, la 1 y la 2 están claramente relacionadas y pudieran por tanto escribirse dentro de un mismo programa, ya que utilizarán mutuamente las mismas rutinas para generación del listín. Las tareas enunciadas como 3 y 4 también están relacionadas en cuanto que para ser capaces de revisar una ficha necesitamos primero elegir y observar dicha ficha. Es aparente por tanto que lo más apropiado es dividir la aplicación en dos programas separados, uno tratando con la creación del listín telefónico y el otro tratando con el uso cotidiano del mismo. Las ventajas son numerosas; siendo la principal la propia sencillez involucrada. Manteniendo separadas las diversas labores a efectuar por el operador, la confección del programa es más fácil y más simple de realizar, y además al utilizar el listín el tamaño reducido del programa que ha de tenerse en memoria en cualquier momento significa que el tiempo empleado para traerlo y ponerlo en ejecución se verá grandemente reducido. Uno de los elementos frustrantes del almacenamiento en cinta cassette es el tiempo empleado en traer el programa del cassette y cargarlo en la memoria, incluso con la posibilidad que tiene el CPC 664/464 para adoptar dos **velocidades de escritura** mediante el comando de clave **SPEED WRITE.** Por lo tanto cualquier ayuda de programación que acorte ese tiempo será una ventaja. Eso obviamente no ha de ser tenido en cuenta cuando se está usando almacenamiento en disquete. Una vez que se ha tomado la decisión de desarrollar una serie de programas para una aplicación deben precederse de una breve rutina que actúe como menú principal para que el operador elija cúal es el programa que se requiere. La misma subrutina puede usarse además, como 'página de títulos' para presentar las funciones de la aplicación y cualquier información esencial que asegure no se eligen opciones incorrectas.

Los programas por tanto que confeccionaremos para esta aplicación del listín telefónico serán y son:

1) Menú de opciones - En la pantalla se muestran las opciones para elegir cual es el programa productivo apropiado.

2) Genere - Programa para preparar y almacenar el listín telefónico.

3) Utiliz - Programa para usar el directorio día a día.

Planificando las imágenes en pantalla

El menú de opciones será el que primero veremos. El plan usual es producir un pequeño dibujo ilustrativo de la página de títulos seguido de la presentación de las instrucciones esenciales para poner en marcha las facilidades ofrecidas.

Imágenes en pantalla para la aplicación del listín telefónico

1) Página de títulos para el menú de opciones:

```
Listin Telefonico
 Listin Telefonico
  Listin Telefonico
   Listin Telefonico
    Listin Telefonico
     Listin Telefonico
      Listin Telefonico
       Listin Telefonico
        Listin Telefonico
         Listin Telefonico
          Listin Telefonico
           Listin Telefonico
            Listin Telefonico
             Listin Telefonico
              Listin Telefonico
               Listin Telefonico
                Listin Telefonico
                 Listin Telefonico
                  Listin Telefonico
                   Listin Telefonico
                    Listin Telefonico
                     Listin Telefonico
```

Figura 7.1: Imagen inicial de la parte primera: menú de opciones

2) Información esencial para poner en acción cada una de las facilidades del programa:

Que deseas hacer?

<A> generar o agregar datos

<B> utilizar el listin

Pulsa la <letra> pertinente

Recuerda que debes haber creado

el listin antes de utilizarlo.

Figura 7.2: Las opciones e instrucciones iniciales

3) Generar o agregar nombres y números al listín telefónico, un 'submenú' inicial para la parte segunda de la serie de programas:

Elige el utensilio que requieras:

<C>reador de un Nuevo listin

<A>gregador de datos al listin

Pulsa <C> o <A>

Figura 7.3: La parte segunda ofrece dos alternativas - esta imagen en pantalla proporciona los detalles

4) Imagen para la introducción de datos agregándolos al listín telefónico:

```
INscribe los datos

Nombre        Apellidos     Telefono



Cantidad de fichas:  1
```

Figura 7.4: La parte segunda permite al usuario inscribir nombres y números del listín telefónico

5) El 'submenú' inicial para las facilidades disponibles en la parte tercera de esta serie de programas:

```
Facilidades Disponibles

Mandame cada accion pulsando su numero
y dime que ya VALE pulsando <ENTER>

<1> Repase el listin

<2> Busque un telefono

<3> Enmiende una ficha

<4> Termine este 'curro'


?
```

Figura 7.5: La parte tercer ofrece cuatro opciones -esta imagen de pantalla muestra dichas opciones

Secuencia, selección y repetición

En este momento, vamos a dejar la aplicación que estamos considerando y vamos a pasar a presentar alguna teoría imprescindible que vamos a requerir. Es ahora prudente observar tres términos que pueden aplicarse a virtualmente cada acción de cualquier máquina o animal viviente que lleve a cabo ciertas **tareas** con el fin de alcanzar metas específicas.

Primero los describiremos, y luego explicaremos las maneras en que pueden aplicarse esos términos no sólo al programar el CPC 664/464, sino que además los aplicaremos a las tareas llevadas a cabo en nuestra vida cotidiana.

Secuencia - Una tarea/labor específica es meramente llevada a cabo y el que efectúa la tarea pasa luego a la siguiente.

Selección - Una tarea/labor ha de cumplimentarse, pero sólo si se satisface una determinada condición antes de iniciar la tarea. Por ejemplo, se propone una pregunta que exige una respuesta si/no. Una respuesta 'si' hará que como resultado se efectúe una tarea; una respuesta 'no' dará como resultado que se efectúe otra tarea distinta.

Repetición - Una única tarea/labor, o una serie de tareas/labores, que pueden ser una tarea en forma de secuencia o de selección, ha de **reiterarse** hasta que cumple una determinada condición. Por ejemplo, una carretilla ha de llenarse con tierra usando un cubo y una pala; el cubo se llenará repetidamente con arena y se vaciará en la carretilla hasta que la tierra en la carretilla alcance un nivel predeterminado.

Vamos a examinar ahora como estas categorías teóricas de tareas nos ayudan en la propia tarea de confeccionar un programa.

Secuencias de tareas tal y como se aplican en la vida diaria y en la creación de un programa

El concepto de colocar cada actividad en **secuencia**, es decir 'en **procesión**' o **consecutivamente** una tras otra, es fundamental en la manera en que la mayoría de la gente organiza sus actividades cotidianas. En esencia, si un cierto número de tareas se colocan en secuencia, se establece un orden lógico para la realización de las mismas. Por ejemplo no puedes hacer la cama antes de que te levantes de ella.

No puedes ver la televisión hasta que la hayas encendido. De igual manera, en lo que se refiere a usar un programa de aplicación en un ordenador, tal como el listín telefónico, no puedes usarlo para consultar el teléfono de alguien hasta que realmente hayas pasado por el proceso de generar el listín e inscribir en él ese dato. En lo que se refiere a la relación entre el confeccionador del programa y el usuario del mismo: la imagen en pantalla que proporciona las instrucciones debe proyectarse antes de solicitar una respuesta del usuario. Por lo tanto, volviendo de nuevo a los comentarios del capítulo 6, la secuencia de 'exposición en pantalla/espera y recogida de respuesta/tratamiento de la respuesta' es una secuencia lógica de acciones que no tendría ningún sentido si se ejecutará en cualquier otro orden.

Selección de tareas tal y como se aplica en la vida cotidiana

La secuencias se convierten en poco realistas en cuanto que presuponen que no hay ninguna interacción con la situación actual, simplemente obedeciendo la misma serie de instrucciones cada vez que se pasa a realizar esa particular secuencia de acciones.

Manteniendo la misma estructura básica, pero incluyendo una pregunta de manera que las acciones subsecuentes queden **condicionadas** por la respuesta a dicha pregunta, nos permite apelar a una serie de opciones y por tanto a un cierto número de posibles 'avenidas a explorar'. La **selección** se logra por el proceso de constatar si se satisface una determinada condición conocida. Entonces, y solo entonces, se llevará a cabo una particular secuencia de acciones. Como una alternativa a la premisa condicional a=b el curso de acción puede basarse en otras diversas estructuras condicionales:

1) **Si** A es menor que o mayor que B **entonces** se efectúa esta secuencia de operaciones o actividades.

2) **Si** A es distinta de B **entonces** se efectúa esta tarea, secuencia de acciones u operaciones.

3) **Si** se requiere una mezcla de condiciones, e.g. igual a, distinto de, menor o mayor que, puede considerarse el uso de palabras clave **conectivas** del tipo **y sí** (and), **o si** (OR), y otras 'conjunciones' similares.

4) La conjunción **ilativa 'y si...'**, de clave **AND**, significa que **todas y cada una** de las condiciones que se mencionan han de satisfacerse con el fin de que la condición compleja resultante sea considerada como **cierta** y por tanto pase a efectuarse la tarea o serie de actividades indicada.

5) La conjunción **disyuntiva, o si...**, de clave **OR**, significa que **basta con que** una de las condiciones mencionadas sea satisfecha, para que la condición compleja resultante sea considerada como **cierta** y por tanto pase a efectuarse la tarea o serie de actividades indicada.

Repetición de tareas tal y como se aplica en la vida cotidiana

En nuestra vida diaria claramente repetimos tareas **similares**, cada hora, cada día, semanalmente, mensualmente, y es bastante posible que anualmente. Por lo tanto hablar aisladamente de **secuencias de actividades**, incluso aunque incluyan premisas condicionales, sin el concepto de **reiterar** cualquier tarea o secuencia de operaciones, no sería realista. Esencialmente, las tareas han de repetirse **mientras que** una condición dada no se vea satisfecha; por ejemplo: la mayoría de la gente va a la escuela cada día de la semana (exceptuando las vacaciones) cuando su edad está entre los 5 y los 16 años. Esto forma una secuencia de operaciones que se lleva a cabo repetidamente **hasta que** se satisface una determinada condición. La forma opuesta de repetición también es válida y debe ser considerada, a saber, la de repetir una tarea concreta **mientras que** se vea satisfecha una condición dada. Un ejemplo sería la instrucción que mande que mientras esté lloviendo y mientras que estés fuera de casa lleves una gabardina. Un ejemplo de programación podría ser que **mientras que** una variable contadora no alcanza un deteminado valor la pantalla cambiaría a cada paso de color.

Las repeticiones o **reiteraciones,** o si prefieres hablar en términos informáticos, los **bucles** preconfinados o condicionados, pudieran ser eternas si el estado necesario para salir del bucle no se viera nunca satisfecho. Claramente estos bucles repetirían la secuencia de tareas ininterrumpidamente o hasta que apagáramos el ordenador. En la mayoría de los casos esta situación no es deseable en un programa y debiera considerarse cuidadosamente antes de incluir un **bucle 'sin fin'** dentro de un programa.

Estructuración natural de los programas de ordenador

A estas alturas debiéramos estar considerando todos y cada uno de los programas **desglosados** en pequeños párrafos que llevan a cabo **tareas** específicas, por ejemplo:

1) Presentador de pantalla de información.

2) Recogedor de la respuesta del usuario.

3) Actuadores subscuente a una respuesta dada.

Cada bloque por sí mismo se construye a partir de una secuencia de operaciones colocadas según el orden de razonamiento lógico. Para aplicar ese bloque o párrafo a la situación corriente se **implementa** con repecto al cumplimiento de diversas condiciones que determinarán el curso real de la secuencia de operaciones y los puntos donde dichas secuencias se repetirán o terminarán.

Cada bloque también formará parte de una estructura mucho mayor a la que puede a su vez darse una estructura similar de secuencias de párrafos que, como resultado de las condiciones implicadas, seguirán un diferente curso de acción de acuerdo con las respuesta del usuario, además de repetir diversas combinaciones de bloques de programa para cumplir las metas establecidas en el propio programa.

Capítulo Ocho

Los Diagramas Clarifican las Cosas

Cuando pensamos en expertos informáticos usando diagramas para ilustrar sus programas viene a la mente el término de **diagrama de flujo** (flow chart en inglés) e inmediatamente nos imaginamos figuras cuadradas, rectangulares y romboidales, con palabras escritas dentro de ellas unidas unas a otras por madejas de líneas que dirigen al usuario hacia arriba, hacia abajo y a través y alrededor del programa. Hay una preocupación muy definida en cuanto a que realmente no son muy lógicos en su forma final.

Aquí, yo voy a sugerir un enfoque alternativo para trazar un diagrama que pueda seguirse desde la parte superior de la página hacia abajo y hacia la derecha exactamente como si se estuviera leyendo una página de texto. Este enfoque de **arriba hacia abajo** en la programación, es una técnica muy popular en estos momentos y esta forma de diagrama encaja con los requisitos necesarios para tal enfoque.

Examina la Fig. 8.1 e intenta identificar a) tareas de la categoría **secuencia,** b) tareas de la categoría **selección,** y c) tareas de la categoría **repetición.** Comienza la lectura en el óvalo de comienzo, sigue el curso descendente de la línea vertical hasta que te encuentres con una línea horizontal, y sigue entonces las instrucciones dadas moviéndote horizontalmente. Al concluir la línea horizontal viaja de nuevo a la última línea vertical y continua tu curso hacia abajo hasta que aparezca la siguiente línea horizontal de instrucciones. Sigue a lo ancho y hacia abajo hasta que todas las instrucciones hayan sido ejecutadas, dependiendo de si se han satisfecho las condiciones establecidas y las respuestas a las decisiones que el usuario haya hecho.

La Fig. 8.1 ilustra el potencial de estos diagramas de flujo de arriba hacia abajo. Observa que es posible utilizar este enfoque para los eventos diarios al igual que para la programación.

Vamos ahora a desglosar los diversos elementos diagramales examinando las siguientes cuatro figuras, las 8.2, 8.3, 8.4 y 8.5.

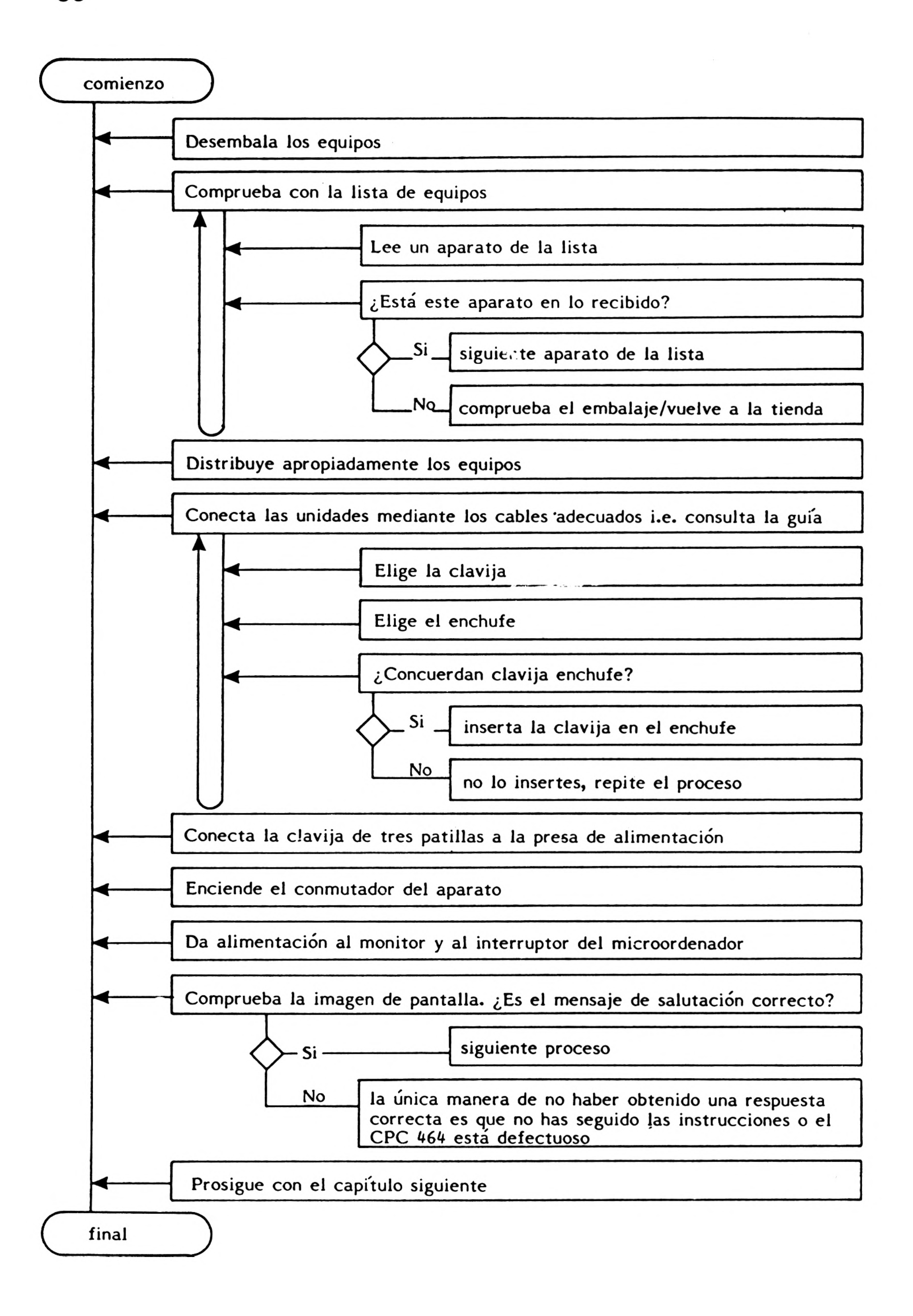


Figura 8.1: Usando un diagrama estructural para preparar tu CPC 664/464

Secuencia de operaciones

La forma de ilustrar un **tramo procesivo** en un programa es muy similar a la manera en que leemos la prosa de un libro. Haz esta operación, luego esta operación, y así sucesivamente hasta completar la tarea. Por eso se denomina **'procesión'** o **secuencia** de operaciones. No hay que tener en cuenta las variaciones que puede haber en las condiciones en un momento concreto. Se supone que estas tareas procesivas se completan de la misma manera todas y cada una de las veces en que se efectúan. Estos diagramas son en esencia resúmenes de acciones. No será hasta que comencemos a usar los términos propios de la programación hasta que apreciemos la importancia de estos tramos procesivos en un programa, por ejemplo con los comandos en modo directo de la Fig. 8.6.

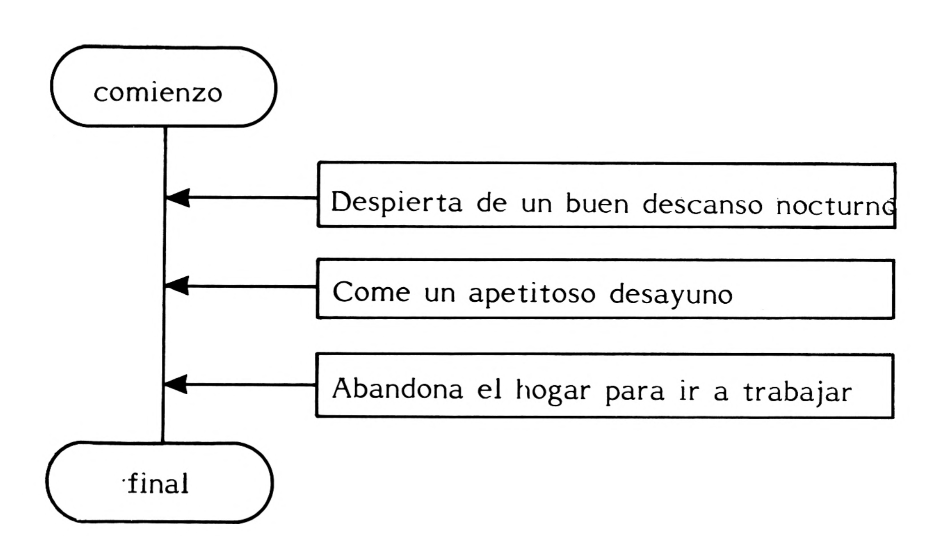


Figura 8.2: Ilustración de un tramo **procesivo,** arreglado de acuerdo con un orden de razonamiento lógico

Selección de operaciones

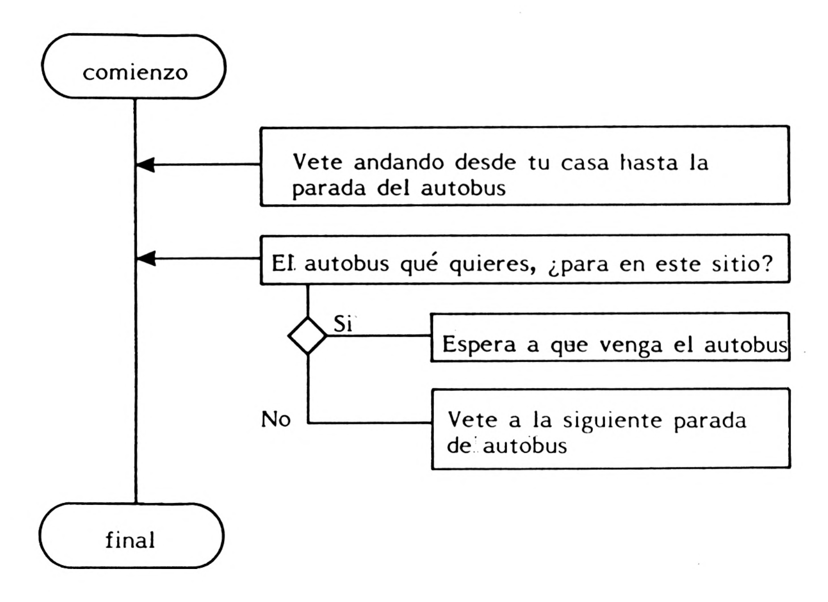


Figura 8.3: La explicación de una **procesión** de operaciones se hace más realista si en ella incorporamos la capacidad para condicionar esas operaciones. La selección de uno o más tramos dentro del curso del programa puede representarse fácilmente por esta clase de diagrama.

Proporcionar una **elección** significa que el usuario ha de adoptar una u otra de dos rutas de acuerdo con la opción que haya elegido. En este ejemplo, la Fig. 8, la tarea queda concluida volviendo a la línea vertical siendo la siguiente instrucción a cumplimentar la que marca el final de la tarea. Si consideramos ahora esta proposición cuidadosamente vemos que no es realista: ¿qué sucedería si la siguiente parada del autobus tampoco fuera aquella donde para el autobus que necesitamos para ir a trabajar? Es dolorosamente patente que el segmento selectivo de la tarea tendrá que poder ser repetido hasta que se llegue a una parada de autobus que sea la adecuada para que la persona pueda ir a trabajar. La línea en forma de U de la Fig. 8.4 demuestra como se incluye esta tarea repetitiva para proporcionar un toque de realismo adicional.

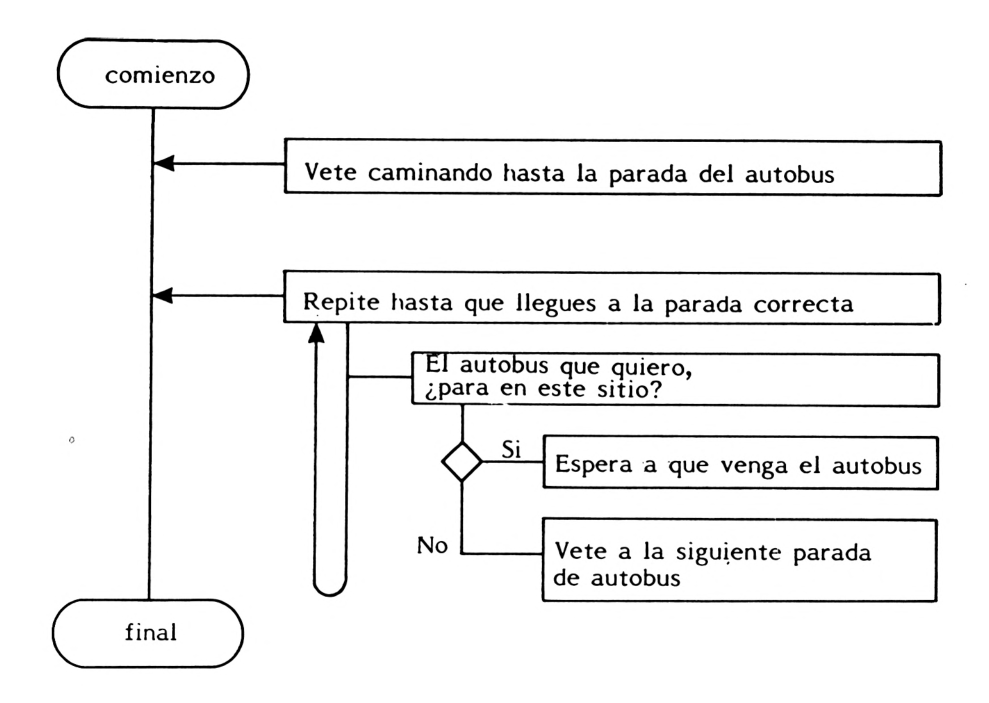


Figura 8.4: Reiteración de acciones incluyendo un tramo **selectivo**

Desarrollo del concepto de iteración

Si se necesita un cierto resultado antes de proceder con el siguiente tramo del programa, se requiere un método de reiterar o repetir la tarea. La primera serie de operaciones en la Fig. 8.5 demuestra este aspecto.

Este ejemplo ilustra el concepto de una selección múltiple con varias opciones repitiéndose a lo largo de la vida cotidiana en diversos momentos, siendo una u otra seguida por una serie o procesión de operaciones. Es en este punto en que el uso de determinados símbolos puede permitir el desarrollo de los detalles adicionales de la subrutinas. El símbolo dice al usuario que parte de la operación será documentada en algún otro lugar, que estará identificado por el número correspondiente.

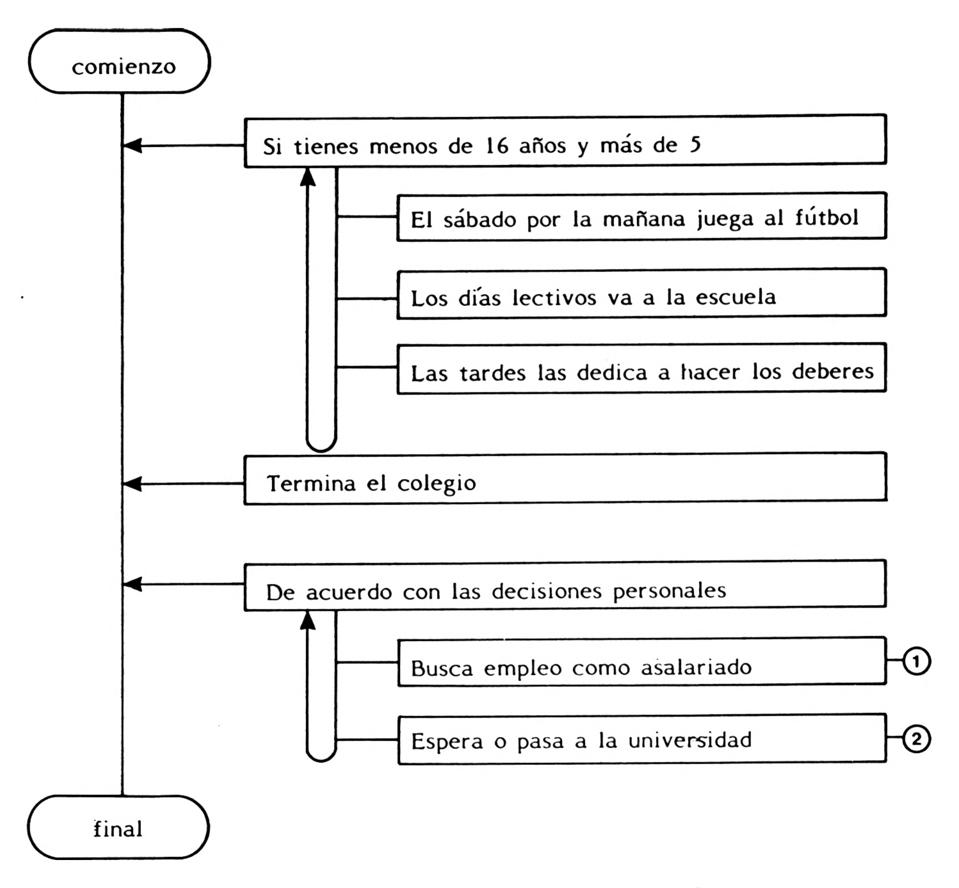


Figura 8.5: Añadiendo una tercera dimensión a un tramo iterativo, la posibilidad de producir análisis detallados y realistas de las series de operaciones se convierte en un sencillo ejercicio logístico.

Flujo de control

Estamos ahora en la etapa en que necesitamos armonizar el enfoque del diagrama y aplicarlo a las realidades de teclear el programa en el CPC 664/464. En los términos del diagrama, el flujo del control guarda relación con las líneas verticales y horizontales que han de ser seguidas para llegar a concluir con éxito una tarea específica. Cuando tratamos de confeccionar un programa de ordenador, como se mencionó en el capítulo 2, el flujo de control queda gobernado por la sección principal situada a la cabeza del programa, y terminada por el comando para que **FINALICE**. Esta sección determina el orden en que se **apelará** a cada una de las subrutinas para llevar a cabo las tareas específicas.

Uso de bloques de programa-subrutinas

Identificando todas y cada una de las subrutinas encargadas de llevar a cabo una tarea, dibujando luego el diagrama estructural, acoplado a un diagrama de la sección principal de control del programa, la labor de confeccionar realmente el programa de acuerdo con el lenguaje BASIC del ordenador, queda reducida al ejercicio de aprovechar las posibilidades disponibles con ese lenguaje particular del ordenador que se vaya a usar.

La belleza de un programa de aplicación tal como el listín telefónico es que cada posibilidad disponible al usuario del programa encaja netamente en una subrutina con la adición de subrutinas para el menú de opciones y para elementos tales como el mensaje 'pulsa cualquier tecla'. Tómate ahora algún tiempo en considerar la creación de diagramas que si fueran estudiados por un usuario del programa safisfaría las metas estipuladas. Asegúrate que puedes hacerlo tanto globalmente en lenguaje llano como detalladamente en el lenguaje de programación BASIC. Si estás 'corto de ideas' toma secciones de la parte tercera del listín telefónico y trata los diagramas con respecto a esas funciones.

El anteproyecto de un programa estructurado

Examina el listado del programa que dimos en el capítulo 2. Serás capaz de identificar cada una de la secciones mostradas en la Fig. 8.6. Usalo como un anteproyecto para desarrollar primero una serie de diagramas estructurales que correspondan al programa de aplicación. Finalmente esos diagramas debieran poderse transferir a las líneas de instrucciones que constituyen el programa, conteniendo todas las posibilidades que se requerirán en la aplicación. También debes comparar los elementos esenciales del programa y deben ser similares a la estructura mostrada.

MEMO: documentación explicando el propósito del programa y otros detalles esenciales.

También pueden incluirse memorandum para describir las variables utilizadas en el programa.

Preparación del CPC464, ya que el microordenador puede requerir ciertos cambios para que tu programa sea eficaz en cuanto al modo, las definiciones de las teclas, los caracteres definibles por el usuario, etc.

Al usar el CPC464 no es necesario declarar previamente el nombre de la variable o asignarle el valor inicial, sin embargo, si el programa va a ser reiteradamente ejecutado puede ser necesario restablecer los valores iniciales de las variables.

La sección de control principal es donde se eligen las diversas subrutinas y se observa el progreso lógico a lo largo del curso del programa. Se puede **recurrir** a cada subrutina varias veces, o solamente una, o meramente como resultado de las respuestas obtenidas del usuario del programa. El control principal puede tener tantas subrutinas a su disposición como el confeccionador del programa requiera.



i.e. **apelación** a subrutina

subrutina 4 - traer y cargar fichero de datos

subrutina 3 - menú de opciones/selección de posibilidad

subrutina 2 - repasar el listín telefónico

subrutina 1 - buscar el teléfono de una persona

```
710 REM elegidor argumento de búsqueda
720 LET nome$="zz":LET apelli$="zz"
730 CLS
740 PRINT"Por qué quieres que busque"
750 PRINT:PRINT" A por nombre propio"
```

Figura 8.6: El anteproyecto de un programa estructurado

Capítulo Nueve

Implementación del Plan: 1

Usando como modelo la aplicación del listín telefónico, estamos ahora en situación de comprender claramente las metas y objetivos de la aplicación tal y como las vió el confeccionador del programa cuando la idea se formuló por primera vez. Es muy importante ser capaz de **imaginar el producto acabado** antes de embarcarse en el tecleo del programa.

El método más razonable de desarrollar esta serie concreta de programa, es tomando cada programa en el siguiente orden:

1) La parte de creación de listín incluyendo la de agregar datos a un listín previamente creado, i.e. parte segunda de la aplicación.

2) La parte de utilización del listín, con las facilidades disponibles una vez que el programa de aplicación sea completamente operativo, i.e. parte tercera de la aplicación.

3) El menú de opciones para unir las dos partes principales del programa y producir una página de títulos para la aplicación.

Así se forma una secuencia lógica para enfocar la confección de los programa.

Generación del listín: parte dos de la aplicación

¿Cúales serán las operaciones que constituyan la sección principal de control del programa que vamos a realizar? Para responder a esta cuestión, pongamos en práctica el enfoque de los diagramas desde **arriba hacia abajo** comentados en el capítulo anterior.

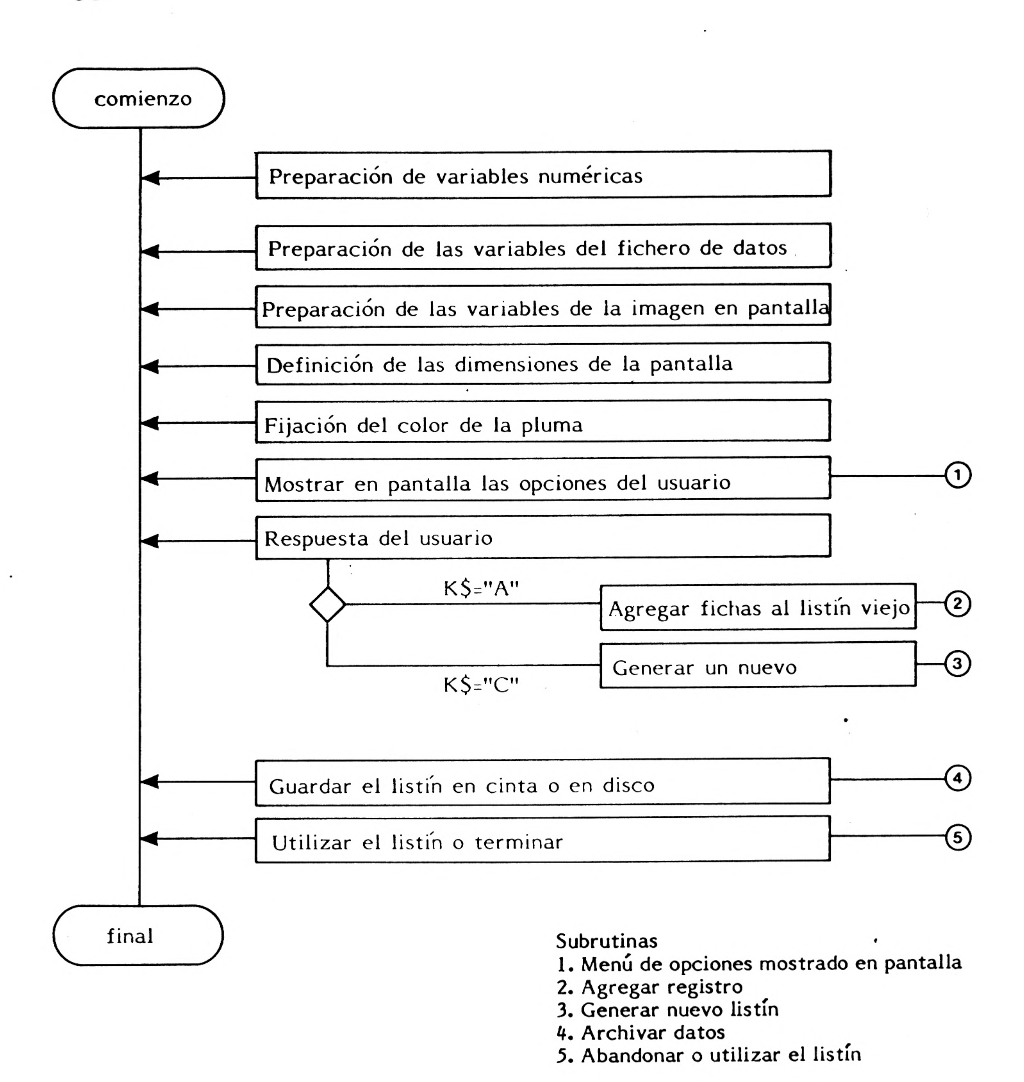


Figura 9.1: Control principal del programa de generación parte dos del listín

El diagrama en la Fig. 9.1 debiera ser ahora autoexplicativo. Los números encerrados en los círculos guardan relación con las subrutinas que se ilustran en las Fig. 9.2 a 9.5. Cada subrutina lleva a cabo una tarea claramente estructurada y virtualmente autocontenida, que cuando sea mezclada con el programa global trabajará como un engranaje en una maquinaria mayor. La relación entre el bloque de control principal del programa y la subrutinas puede asimilarse mucho a considerar que las subrutinas forman un mazo de cartas y el bloque de control principal es el jugador que **usa estas cartas** de manera sagaz para obtener una jugada ganadora.

Enlaza los diagramas de las Fig. 9.1 a 9.6 con los listados del programa mostrados en el capítulo 2. Recuerda que los diagramas son la primera etapa en la confección de un programa a partir de la cual cabe entender el listado del programa.

El bloque de control principal del programa, tal y como se mencionó anteriormente, es el gobernante de todo el edificio del programa. Así puede convertirse en una mera serie de **apelaciones a la subrutina** para que entre en acción con el fin de asegurar el resultado final deseado, i.e. el objetivo estipulado para el programa. Sin embargo, en el ejemplo que estamos considerando hay dos rutas que el usuario puede adoptar según la opción que elija A o C como contestación a lo ofrecido en la primera subrutina. Como resultado, el programa coloca la letra pertinente como contenido de la variable que se ha denominado K$.

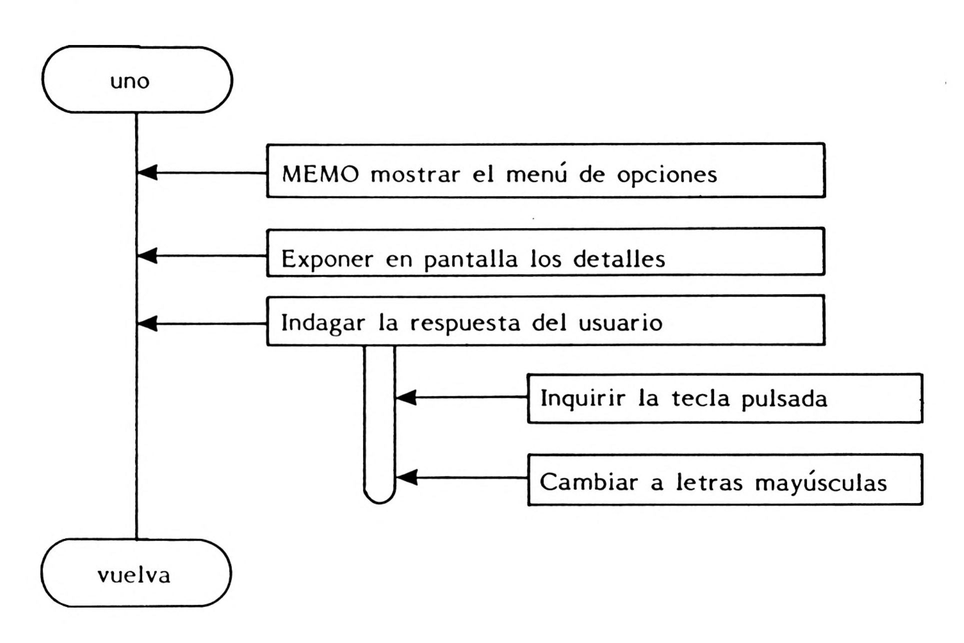


Figura 9.2: Subrutina 1: imagen en pantalla y opciones del usuario

El control del programa revierte luego al elemento principal de control una vez más, donde o bien **se cita** -poniéndola en acción- la segunda o la tercera rutina como resultado de lo elegido por el usuario entre las opciones disponibles. Al concluir una u otra de estas subrutinas el control vuelve a revertir al bloque principal de control del programa.

Se emplea una sutil técnica de programación para disminuir el tiempo empleado en teclear las instrucciones del programa. Observa que la subrutina 2 simplemente **trae** de la cinta o del disco el listín previamente creado, preparando las variables relevantes que controlan la cantidad de fichas archivadas, etc. para que su valor corresponda con el apropiado. Cambiando el contenido de K$ a la letra C da como resultado la ejecución automática de la subrutina 3, porque una vez que la tarea de traer el listín telefónico previamente generado haya sido llevada a cabo el programa se escribe de manera que tanto la agregación como la generación de un listín enteramente nuevo usan las mismas instrucciones del programa, i.e. la subrutina 3.

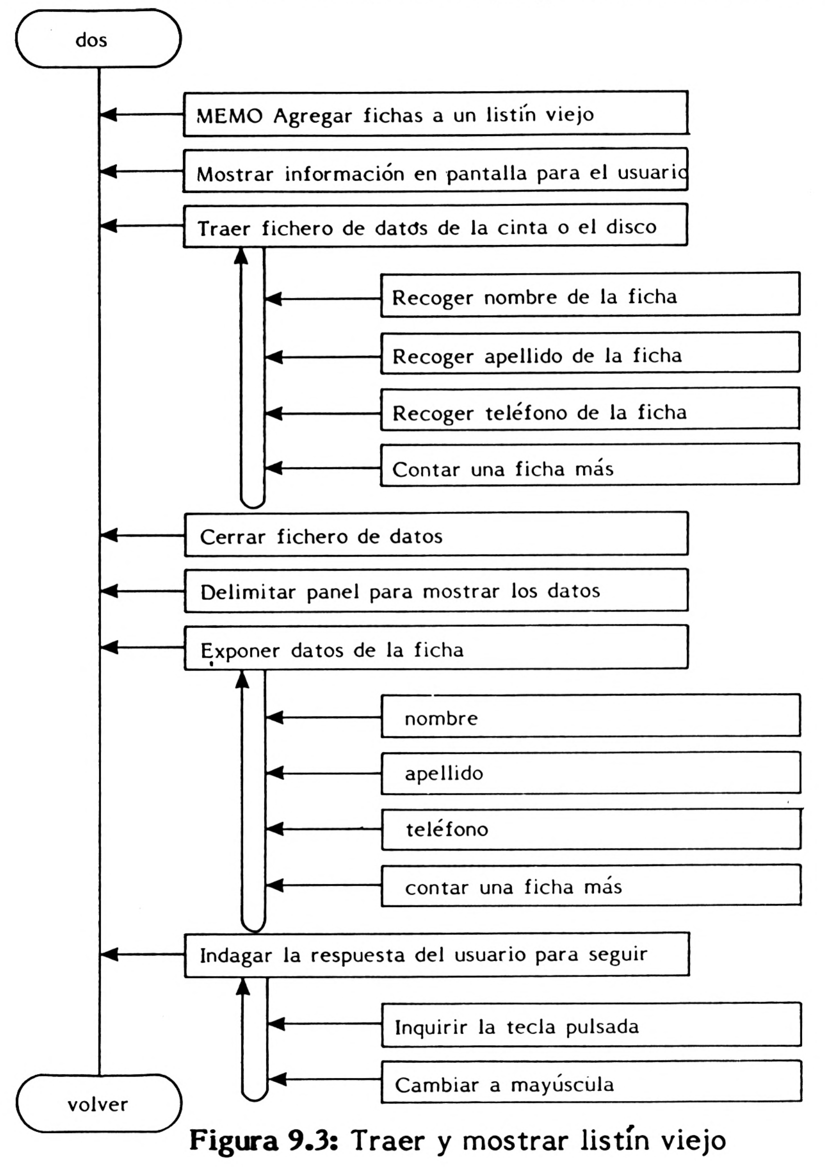


Figura 9.3: Traer y mostrar listín viejo

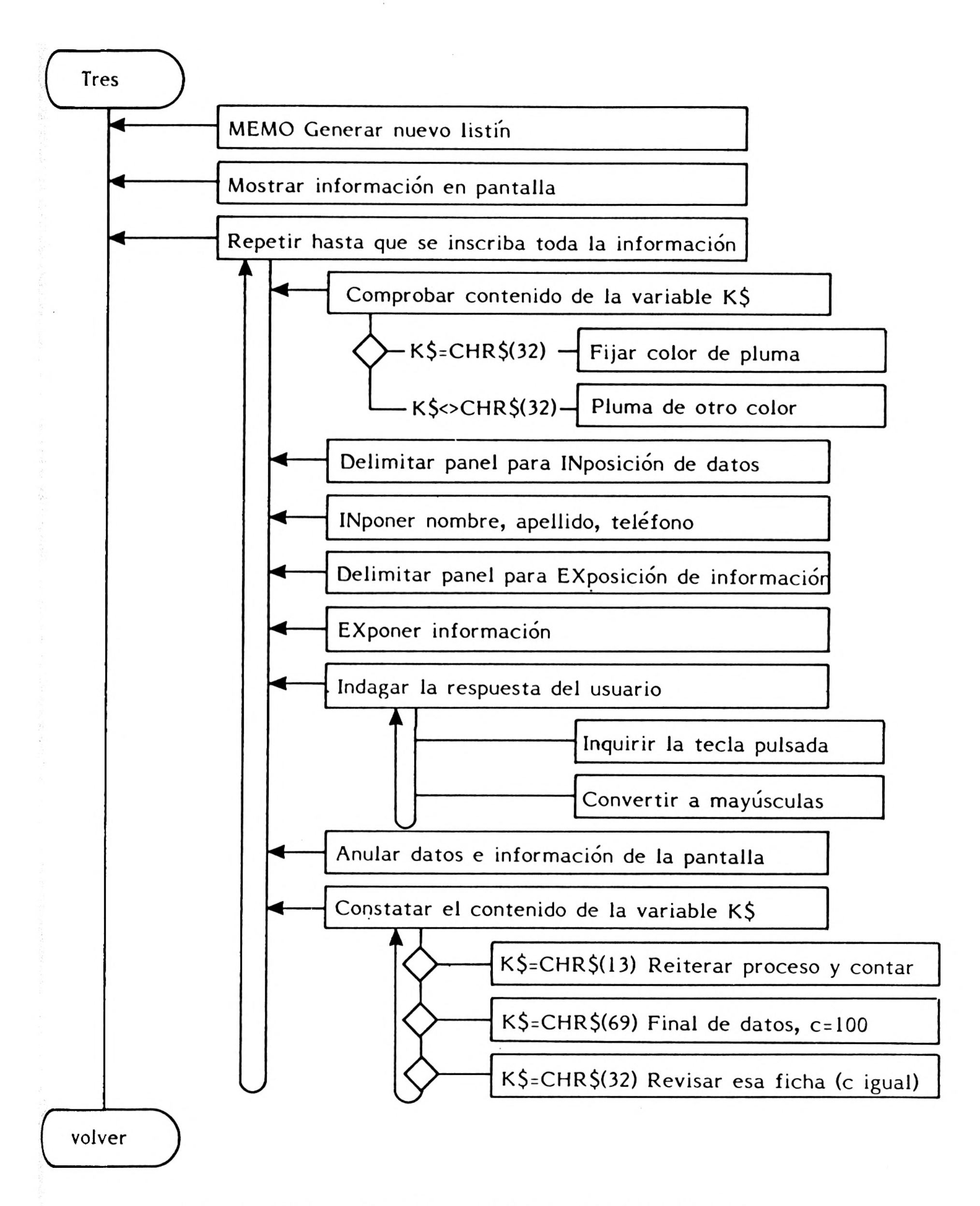


Figura 9.4: Subrutina 3: Inscribir los datos de una ficha del listín

Al concluir la tarea el listín nuevo o con los datos añadidos, ha de **guardarse** sobre cinta o sobre disco (subrutina 4) y dar al usuario la opción de finalizar o de traer de la cinta o del disco el siguiente programa de la serie (parte tercera) para aprovechar inmediatamente el listín, i.e. subrutina 5.

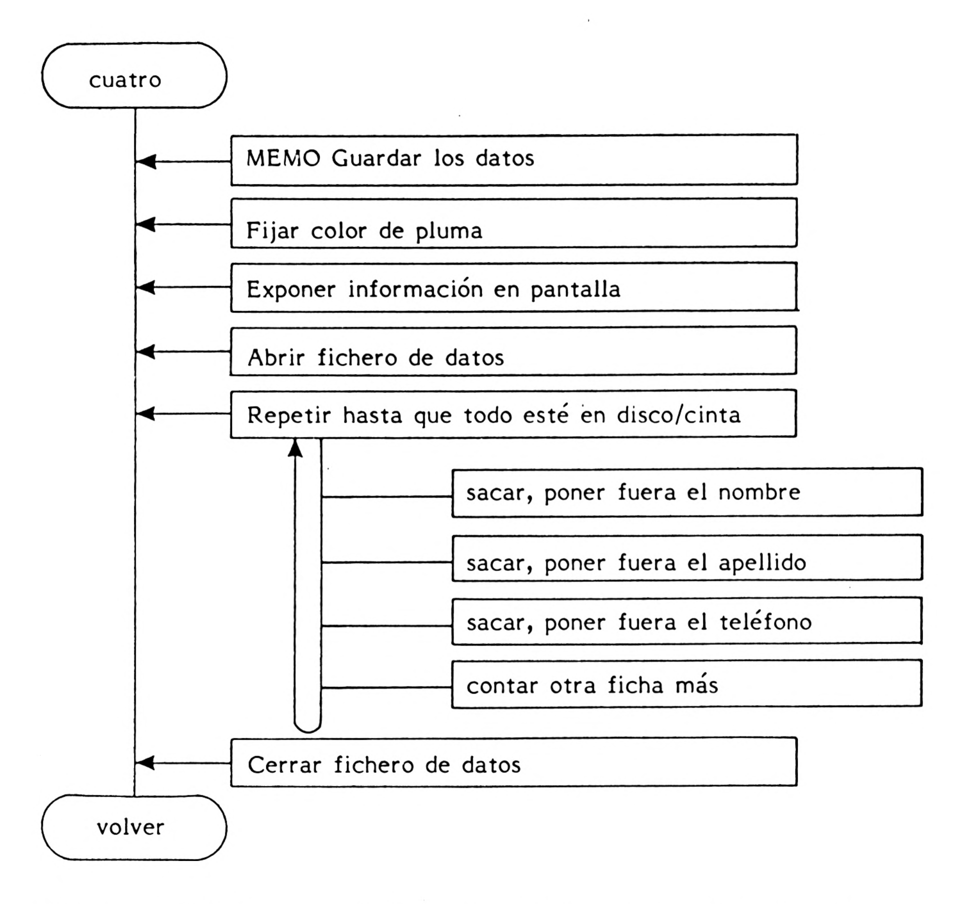


Figura 9.5: Subrutina 4: Guardar en cinta o en disco los datos inscritos

Comentarios sobre las subrutinas de la parte segunda

Cada diagrama guarda una clara relación con una tarea específica de la aplicación y debiera ser fácilmente legible si se siguen cada una de las acciones a lo ancho de la página. La **procesión** de acciones debiera ser consecutivamente hacia abajo de la página.

Las primeras dos subrutinas constituyen sencillas **secuencias** de acciones, con un número de operaciones reiterativas que, o bien se terminan porque el usuario hace la respuesta convenida, o bien por la exposición de los datos i.e. nombres y números de teléfono en la pantalla, o trayendo del sistema de archivo en cinta o en disco hacia el ordenador la información pertinente.

El bucle repetitivo, subrutina 1, es un sencillo **bucle condicionado** que permite al programa esperar inquiriendo cúal es la tecla pulsada, con la función de clave **INKEY$ mientras** que no se pulse la adecuada, i.e. el usuario haga su elección.

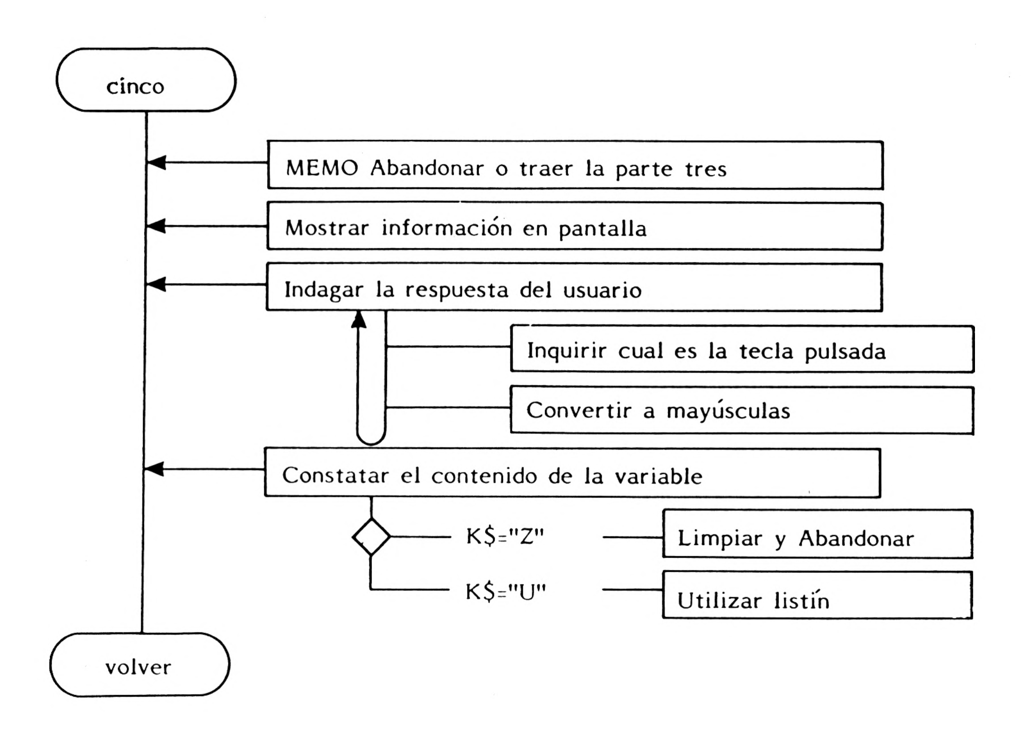


Figura 9.6: Subrutina 5: Abandonar o traer el programa de utilización

Se pueden usar palabras clave del BASIC dentro de un diagrama -aunque no es muy conveniente- si con ello se ayuda eficazmente a demostrar la operación concreta que ha de efectuarse. Por eso es por lo que inquirir la tecla pulsada, con la función de nombre clave **INKEY$**, va generalmente seguido por la conversión a mayúsculas, mediante la función **UPPER$.**

Haz tus diagramas como paso intermedio entre los bosquejos de la imagen en pantalla y las propias líneas de instrucciones del programa. Varias líneas de instrucciones pudieran ser identificables pero estar inscrustradas en frases y palabras que claramente expliquen las acciones a realizar.

Los nombres de variables, como en la subrutina primera, también debe usarse con restricciones, pero aprovecharse para demostrar como sucede en el caso de las variables múltiples o colectivas, para las que hay que **ocupar** espacio en memoria de acuerdo con sus **DIMensiones** y que son las utilizadas para recuperar o depositar repetidamente la información en cinta o en disco, o para exponer sus valores en pantalla hasta que todos los datos retenidos dentro de esa variable múltiple hayan sido empleados. Por ejemplo, el segundo bucle reiterativo, subrutina 2, expone cada nombre y cada número de teléfono registrado en la memoria.

La subrutina 3 presenta una complicación adicional en el tramo reiterativo de acciones a realizar. La subrutina queda preparada inicialmente para la exposición de datos en pantalla y todas las operaciones restantes guardan relación con la inclusión de nombres y números de teléfono. Los dos tramos selectivos guardan relación con lo buscado y si los datos son correctos o si se ha terminado la generación del listín. La INposición real de los datos es meramente un tramo procesivo de operaciones que puede acoplarse con diversas distribuciones en pantalla cuando coloquemos las instrucciones del programa.

A través de una etapa de planificación, relaciona las imágenes en pantalla y observa la necesidad de alterar los colores. Si eso se hace realmente cambiando la tinta o cambiando el cursor de texto que se utiliza depende realmente de los gustos individuales o de las manías del programador.

Si encuentras difícil pasar directamente al trazado de los diagramas que incluyen todos los bucles repetitivos y las elecciones a efectuar, comienza enumerando las operaciones a realizar en un diagrama preliminar en la forma de una mera procesión o secuencia de operaciones. Añade a eso un segundo diagrama que complete algunos de los detalles. ¿Cúal es la serie de tareas que ha de repetirse mientras se cumpla una determinada condición? ¿En qué puntos ha de incluirse la rutina para exponer el mensaje de 'pulsar cualquier tecla'? ¿Cuándo estará el ordenador procesando información por sí mismo, tal como cargar datos desde la cinta o el disco? Tales operaciones del ordenador quedan generalmente aseguradas por medio de las instrucciones del programa que constituyan un bucle inscrustado dentro del mismo, en un punto concreto.

Capítulo Diez

Implementación del Plan: 2

Aprovechando la parte tercera de la aplicación de listín telefónico se pueden examinar complicaciones adicionales de la programación y diversas técnicas para señalar las acciones u operaciones a realizar. Brevemente diremos que dos de los métodos más útiles en la programación son primeramente, la inclusión de un bucle en el programa que conste de diversas acciones, cualquiera de las cuales el usuario puede elegir cuando y como desee; y en segundo lugar, la ejecución de una operación si y sólo si ha tenido lugar previamente un determinado suceso o evento.

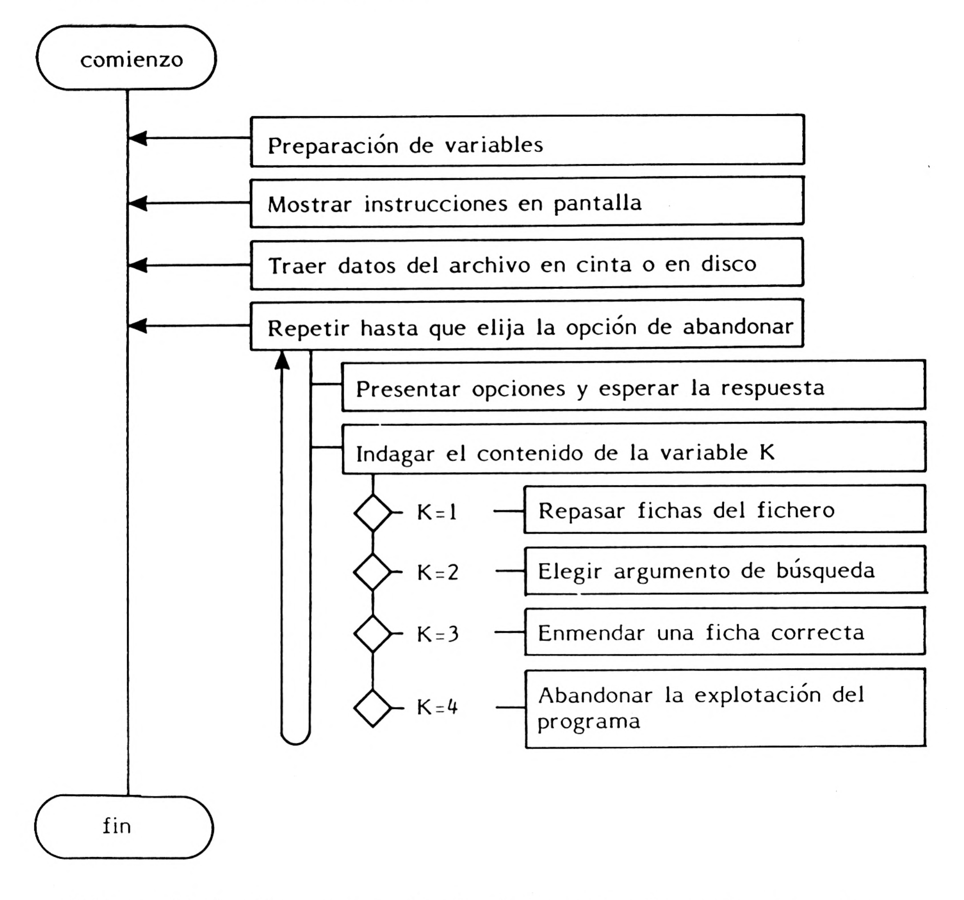


Figura 10.1: Elemento principal de control en el programa de utilización - parte tercera del listín telefónico

Utilizando el listín: parte tercera de la aplicación

La Fig. 10.1 ilustra el elemento principal de control de programa. Las restantes figuras muestran los diagramas de **arriba hacia abajo** de las subrutinas **citadas** dentro del elemento principal de control. Una vez que se hayan preparado las variables y la imagen en pantalla los datos del listín son traidos de la cinta o del disco por una subrutina citada desde el elemento principal de control. Una vez que ésta haya concluido su tarea, se entra en un bucle repetitivo para ofrecer al usuario las facilidades disponibles. La única salida posible de este bucle de instrucciones es cuando el usuario elige la opción de abandonar la explotación del programa, i.e. haciendo que la importantísima variable K tenga como contenido el valor 4. Las líneas relevantes del programa son:

```
110 WHILE K<>4
120   GOSUB 380 :REM a preguntar operacion requerida
130   ON K GOSUB 550,710,950,1230
140   WEND
```

Mientras el programa esté llevando a cabo la tarea de 'Elegir opción' (llevada a cabo por el párrafo comprendido entre las líneas 380 a 540 de la parte tercera), el usuario del programa debe contestar con un número del 1 al 4 ambos inclusive. Esta elección numérica se convierte en el contenido de la variable K. El **desvío** de acuerdo **con** esa variable, mediante la instrucción ON K GOSUB, es una instrucción especial y muy útil del BASIC. La instrucción que le manda **vaya-y-venga-** al acabar a una línea determinada, en el caso de esta instrucción selectiva, puede ir seguida de cualquier cantidad de números de línea que el usuario del programa pueda requerir. Si la variable selectora, en este caso la K, tiene como contenido el valor 1, se desviará el curso del programa hasta la primera subrutina mencionada en la lista; si el valor es 2 se desvía hasta la segunda subrutina de la lista y así sucesivamente. La potencia de esta instrucción es particularmente aprovechable cuando se trata de tomar la acción **acorde con** el valor ingresado para elegir una de las opciones ofrecidas en un menú mostrado en pantalla. Si el contenido de la variable selectora no corresponde a ninguna de las subrutinas mencionadas en la lista, entonces no se **desvía** a ninguna de ellas; por lo que es necesario **anidar** la instrucción de clave **ON... GOSUB...** dentro un bucle repetitivo para garantizar que el programa ejecutará sólo las posibilidades que se le ofrecen al usuario. La elección hecha por el usuario queda así inmediatamente 'validada'.

Cada subrutina ejecutable, al terminar con la instrucción para que **vuelva,** hace que al concluir su actuación **ceda** de nuevo el control al bucle reiterativo del programa permitiendo así efectuar de nuevo otra opción. Si se pulsa un tecla que no sea correcta, el bucle asegura que el programa no termina, sino que se le ofrece otra oportunidad al usuario de elegir.

Vamos ahora a considerar cada una de las subrutinas tal y como aparecen en el listado del programa. La primera subrutina.que entra en acción como consecuencia de ser citada en el elemento principal de control del programa, hace que se **ingresen** los datos que constituyen el listín telefónico específico de cada usuario.

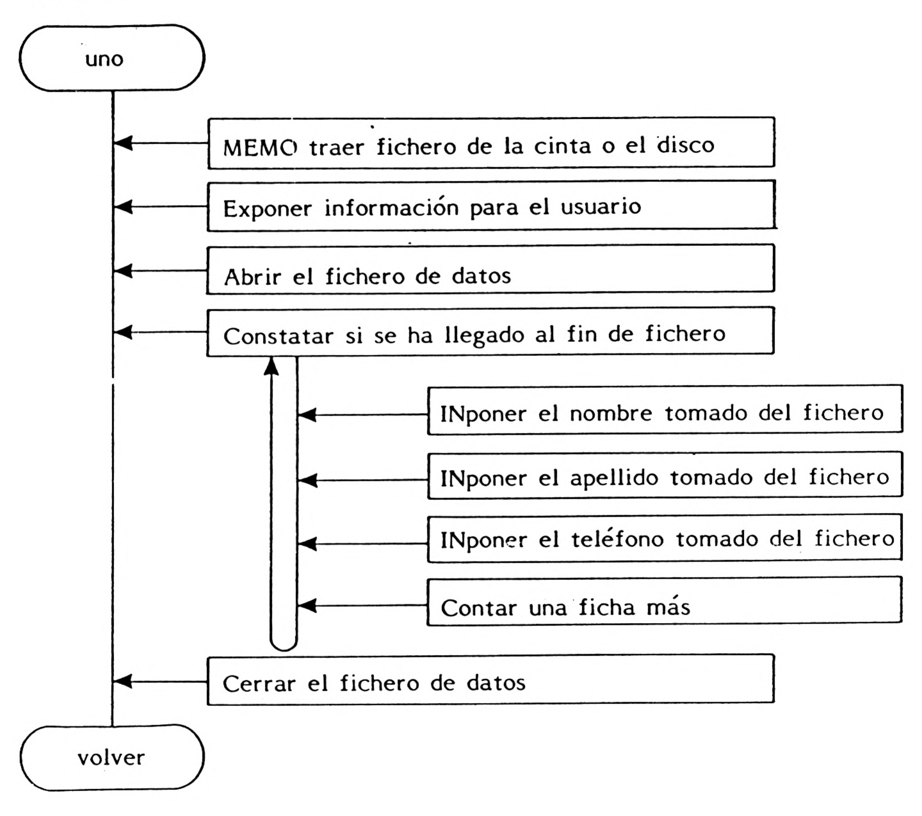


Figura 10.2: Subrutina 1: Trayendo el fichero de datos de la cinta o del disco

Una vez más y como siempre cada acción que el ordenador realiza según las instrucciones que tiene programadas sigue un **procedimiento** predeterminado. Primeramente se anula lo expuesto en pantalla (se 'aclara' la pantalla) y se suministra información al usuario. Luego se cumplimentan las líneas de programa específicas de la tarea a realizar por esa subrutina.

Dado que los datos han de ser traidos de un equipo periférico, i.e. cinta o disco, ha de **abrirse un cauce** especial para la transferencia, hacia el interior del ordenador central, de esa información. Una vez que se ha facultado ese **canal** de comunicaciones, el ordenador comienza la tarea repetitiva de INponer (poner dentro de la memoria) el nombre, el apellido y el número de teléfono que ha tomado del fichero hasta que todos los datos reflejados en el listín hayan quedado registrados en la memoria. El bucle queda entonces terminado porque se ha enviado una **marca** que está grabada en el disco o en la cinta para indicar el **final de un fichero** siendo la palabra clave de esta función en el BASIC la de EOF, que es abreviatura de End Of File. En ese momento el contenido de la variable contadora del programa queda anotado y registrado colocando dicho contenido como valor de la variable denominada **fichas.** Con el fin de hacer que se **cierre el cauce** de comunicaciones previamente abierto, y que era un cauce para INposición o INgreso de datos procedentes de la cinta o del disco, utilizaremos el comando de clave **CLOSEIN** una vez que hayamos recogido todos los datos del fichero. La subrutina ha concluido ahora su tarea y el curso del programa **vuelve** al elemento principal de control con la ventaja de que ahora ya tenemos todos los datos del listín telefónico asentados en la memoria del ordenador.

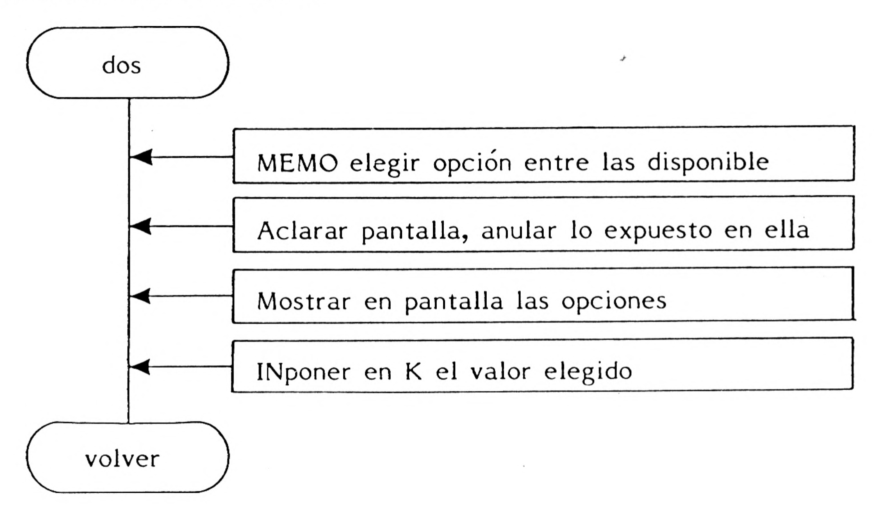


Figura 10.3: Subrutina 2: Presentar las opciones y recoger la respuesta del usuario

Antes de que el usuario pueda realmente hacer su elección, debe suministrársele la información adecuada. La subrutina para elegir la opción disponible proporciona esa información y permite al usuario que opte por una de las posibilidades que se le ofrecen.

Es simplemente una **procesión** de instrucciones para mostrar en pantalla que se ejecutan ordenado y de lógicamente antes de regresar de nuevo al elemento principal de control. La pantalla queda en **claro, se expone** la información en la pantalla, y mediante la instrucción para que se **imponga** el dato tecleado como valor de una variable, se recoge la opción elegida por el usuario. Esta subrutina se repetirá en numerosas ocasiones ya que se **apela** a ella desde un bucle reiterativo incluido en el elemento principal de control de este programa. Si dentro de la subrutina se efectúa una elección incorrecta volverá a repetirse la petición de respuesta sin que ninguna otra subrutina entre en acción; en cambio **si** se hace una elección correcta, i.e. se pulsa las cifras 1 a 4, **entonces** se pasa a ejecutar una de las siguientes subrutinas antes de que vuelva a recurrirse de nuevo a la subrutina de elegir opción.

Las subrutinas ejecutoras de las posibilidades del programa

Repasador del listín telefónico

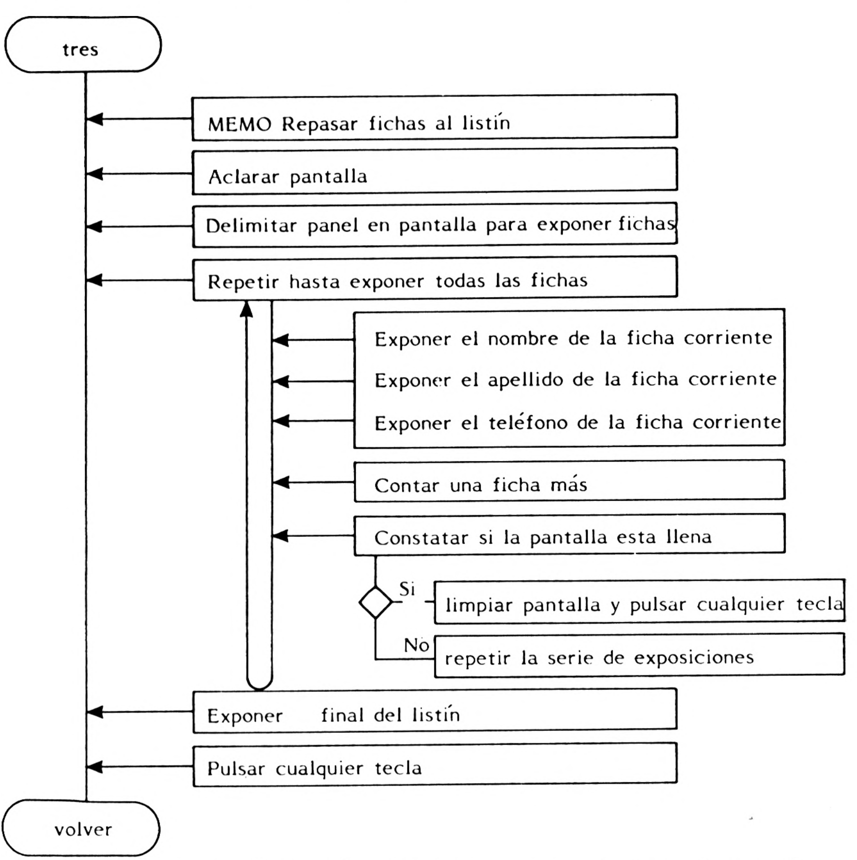


Figura 10.4: Subrutina 3: Repasar las fichas del listín

Esta facilidad de repasar el fichero hace que sucesivamente se expongan en pantalla todos los nombres y números de teléfono inscritos por el usuario en algún momento anterior. Aparecen en el orden que se conoce como **orden de inclusión,** i.e. el mismo orden **cronológico** en que los datos fueron INscritos en el listín.

Si la pantalla fuera lo suficientemente grande como para acomodar en ella todos los datos del listín en una sola 'plana', esta rutina podría simplemente ser una procesión de exposiciones que repetidamente expusiera los datos del listín. Eso no es posible de manera que el programa tiene que efectuar una comprobación para asegurar que solamente se muestra en cada momento **una plana** de información en pantalla. Para pasar a la siguiente plana de información ha de ejecutarse la rutina que expone el mensaje de pulsar cualquier tecla, como se indica en la Fig. 10.8.

El orden lógico es de nuevo anular lo expuesto en pantalla, preparar la imagen en pantalla, exponer repetidamente los números y teléfonos; comprobar mientras tanto si la pantalla está o no llena de datos, y pasar a cumplimentar la operación adecuada. Al concluir este bucle reiterativo mostrar un signo de 'final del listín telefónico' y antes de concluir la subrutina y de devolver el control a la subrutina principal del programa, se apela a la subrutina que presenta el mensaje de 'pulsar cualquier tecla'.

Elección del argumento de búsqueda

También esta subrutina sigue una estructura estándar de exposición de opciones en pantalla, espera a que el usuario elija y luego proceso de la información INpuesta por teclado.

La pantalla se **aclara,** y se presenta en ella información que ayude al usuario a elegir su respuesta. Se da la opción de pulsar una determinada tecla. Como resultado, se pide luego al usuario que la clave de búsqueda sea o bien el apellido o el nombre propio, de acuerdo con la elección inicial. El nombre por el que debe **accesarse** el listín queda impuesto como valor de una de las dos variables llamadas nb$ o ap$. Observa que ambas son similares a las variables a las que hemos hecho referencia en los datos del listín, pero sin los paréntesis como sufijo que contienen aquellas variables que son elementos de una **tabla ristrada** (véase capítulo 14).

Una vez que se haya inscrito el nombre o apellido cuyo teléfono ha de buscarse, se añaden los títulos de los conceptos y el ordenador accesa el fichero comparando cada ficha con el nombre o apellido inscrito por teclado para ver si es el mismo que el reflejado en alguna de las fichas. Eso se logra comparando sucesivamente los datos reflejados en cada ficha una a una y mediante un bucle repetitivo. Si se encuentra que los datos comparados son el mismo, se cumple la condición establecida en la instrucción **si... entonces...** y la variable **testigo** de este hecho queda fijada al valor 1. Eso significa que el mensaje de 'ninguna ficha encontrada' no aparecerá en pantalla. **En los demás casos,** el ordenador expondrá en pantalla los datos relativos a la información pedida. El proceso se repite aprovechando un bucle condicionado **mientras que...,** combinado con una variable contadora que a su vez es comparada con el contenido de la variable **fichas** para ver si se ha recorrido o no todo el fichero.

Una vez que se hayan cotejado todas las fichas del fichero el bucle queda terminado. Se efectúa una comprobación para ver si cualquiera de las fichas comprobadas tenía reflejado el mismo nombre o apellido que se buscaba fijando acordemente el valor de la variable testigo **tf.** Con el fin de terminar la subrutina se **apela a** la encargada de presentar el mensaje de 'pulsar cualquier tecla' para seguir. Cuando el usuario responde a esta petición el curso del programa **vuelve** al elemento principal de control del mismo.

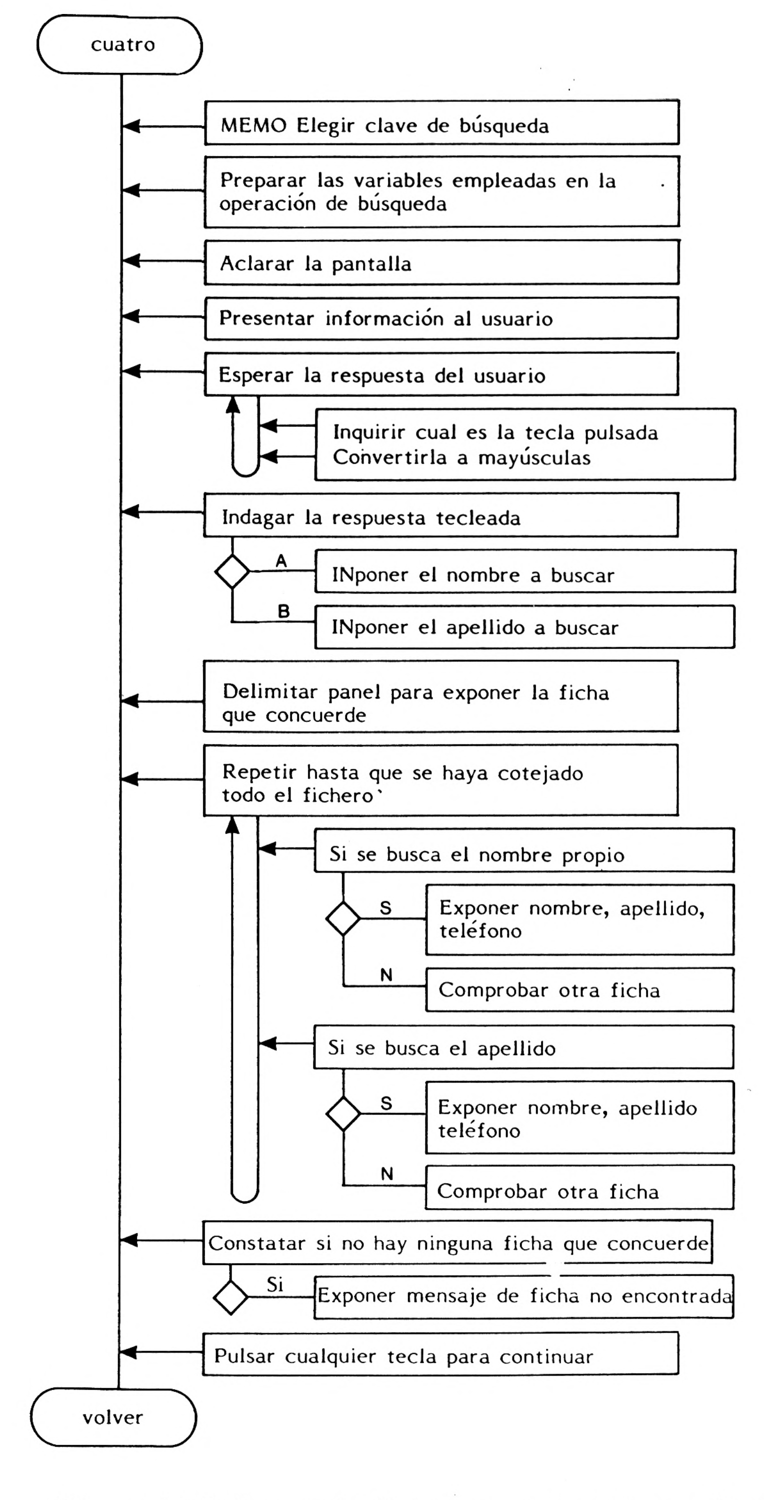


Figura 10.5: Buscar la ficha que concuerde con la dada

Subrutina para enmendar los datos de una ficha

La pantalla se prepara una vez más para la acción subsecuente. La labor que ha de efectuarse se explica mediante un sencillo mensaje. Es imposible corregir los datos de un registro sin que primero se le haya buscado y encontrado, de manera que ¿qué sería más fácil que realmente repetir la ejecución de la subrutina de búsqueda de una ficha? Para evitar exponer en pantalla exactamente el mismo mensaje y hacerlo más en consonancia con la situación, la subrutina comienza en un número de línea mayor que el bloque del programa. El nombre o apellido dado se encuentra o no según corresponda. El control del programa vuelve a la subrutina de enmienda de fichas procedente de la subrutina de búsqueda de ficha. Si no se ha encontrado ninguna ficha la subrutina pudiera muy bien acabar ya que de ninguna manera se puede cambiar una ficha que no existe en el listín. Para continuar supongamos pues que se ha encontrado la ficha que se va a corregir. Cada ficha encontrada tiene **un número de referencia** asociado a ella y en el programa se pide al usuario que IN**ponga** por teclado el número de la ficha que desea corregir. Esa ficha es corregida ejecutando una serie de instrucciones del tipo procesivo, o consecutivo.

El registro hasta este momento sólo está corregido mientras esté almacenado en la memoria del ordenador. Con el fin de conservar la enmienda de forma permanente debemos hacer que el ordenador la **guarde** bien sea en la cinta o en el disco. Esa operación se lleva acabo haciendo que guarde todo el listín completo una vez más y eso es una tarea que consume mucho tiempo en comparación con la de corregir una única ficha. Pero cubrir todas las diversas maneras de manejar ficheros cae desafortunadamente fuera del ámbito de este libro, pero la planificación y el diseño estructural del programa es el mismo.

Las acciones involucradas en archivar datos son: **abrir un cauce** de transferencia para EXponer (poner fuera) los datos, mostrar en pantalla las instrucciones pertinentes, efectuar tareas repetitivas para ir EXponiendo todos y cada uno de los nombres, apellidos y teléfonos -y si hubiera 20 fichas en el listín este bucle sería repetido 20 veces- **cerrar el cauce** de transferencia de salida abierto, y **volver** al elemento principal de control de programa.

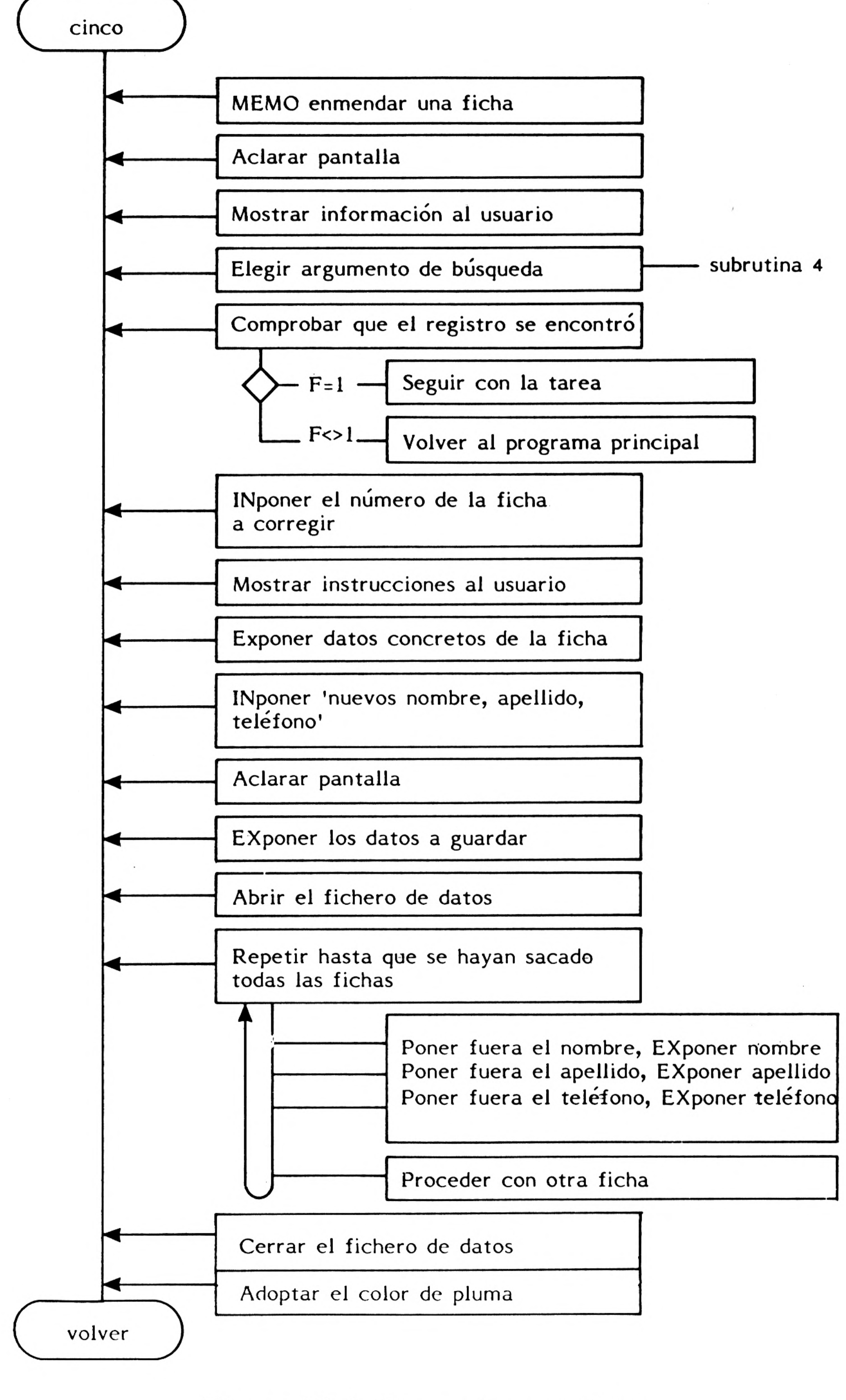


Figura 10.6: Enmendar una ficha

Subrutina para abandonar la explotación

Figura 10.7: Subrutina 6: Abandonar el programa

Esta es una **procesión** o secuencia de acciones directa: se expone un aviso en pantalla y se devuelve el control a la subrutina principal que finaliza todo el programa porque la variable K tiene el contenido de 4. En pantalla queda expuesto el mensaje de despedida.

Subrutina para aviso de pulsar cualquier tecla

Esta subrutina puede operar en combinación con cualquier otra, ya que meramente usa las líneas inferiores de la pantalla. Las líneas del programa hacen que el ordenador espere indefinidamente hasta que el usuario responda, con lo cual la subrutina queda culminada y el control es devuelto al área del programa desde la que se **citó** esta subrutina de 'pulsar cualquier tecla'.

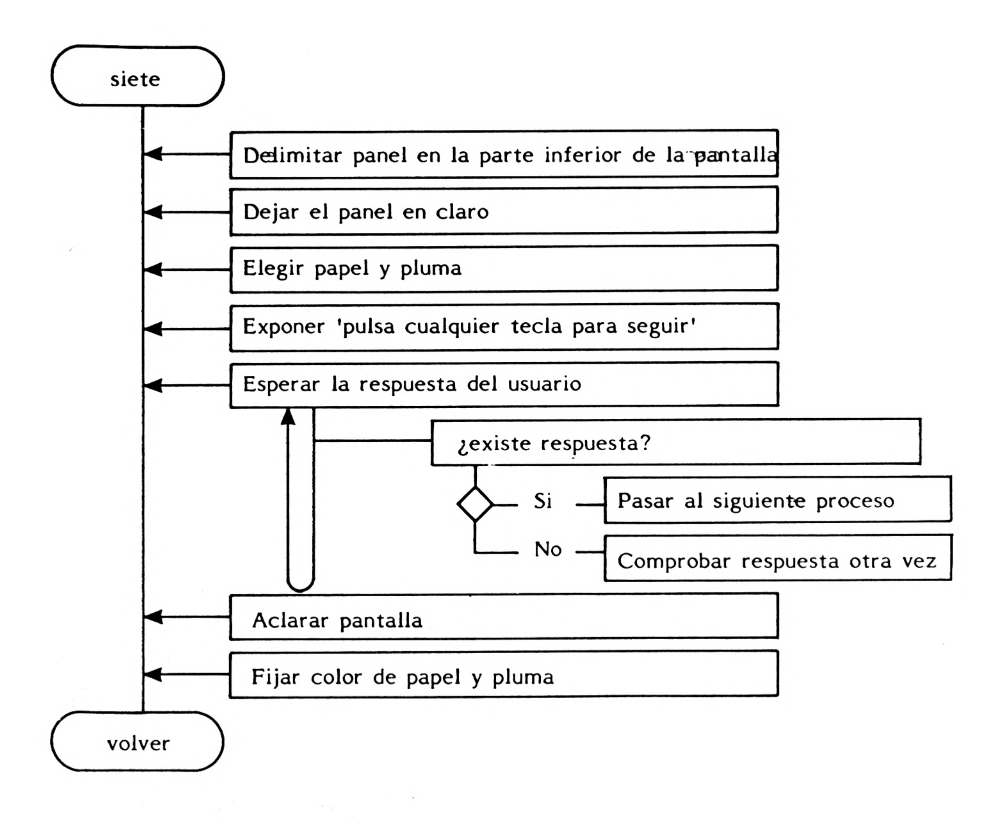


Figura 10.8: Subrutina 7: Mensaje para continuar

El menú de opciones: parte primera de la aplicación

Para completar el capítulo, la Fig. 10.9 ilustra el menú de opciones que permita al usuario elegir entre la generación o la utilización de un listín telefónico. Examínalo e intenta encontrar tu camino a través del diagrama. Luego compáralo con el listado del programa, capítulo 2, o con el propio programa que tendrás en la cinta o en el disco.

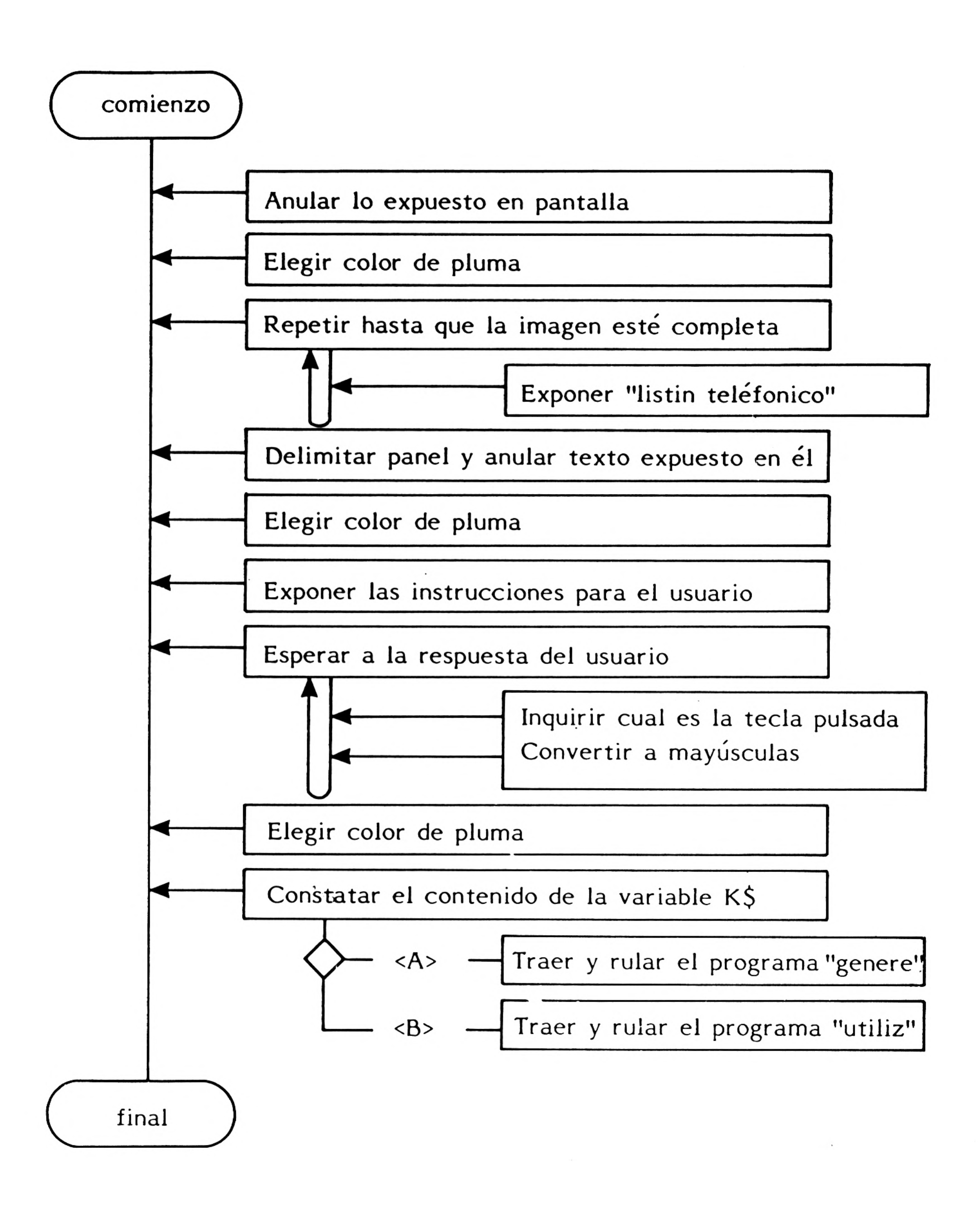


Figura 10.9: Estructura del programa para la parte primera del listín telefónico

Sección E

Tratamiento de Literales - La Clave al Almacenamiento de Información

Capítulo Once

Literales, Constantes y Variables y el Símbolo Dólar ($)

Una **sarta** de caracteres es en definitiva una retahíla de uno o más caracteres alfabéticos **-letras-** o numéricos **-cifras-** o cualquier otro del repertorio **-signos-** que están adosados uno tras otro. Usando **comandos**, es decir trabajando en el modo directo, podemos mandar que exponga en pantalla una sarta de caracteres cualesquiera, como por ejemplo tecleando PRINT "6ab*GO 87". Observa además que cualquier frase es considerada como una mera retahíla de caracteres, y así podremos mandar que exponga "esta frase es una sarta de caracteres", y aprecia que el **espacio en blanco** que separa las palabras 'cuenta' como **otro carácter** cualquiera, ya que ocupa el espacio correspondiente. Eso es un hecho importante cuando estamos incluyendo datos en una base de datos o en un fichero, tal como en nuestro listín telefónico, ya que cada trozo de información realmente es colocado **letra a letra** en una variable literal. Por lo tanto, si el usuario añade un espacio en blanco al final del apellido, la máquina incluirá ese espacio en blanco como valor de la variable literal junto con las letras que intervienen en el apellido. Ensayálo con tu listín telefónico. Luego intenta buscar ese apellido concreto, pero sin incluir ahora el espacio en blanco; -te encontrarás que el programa no puede encontrar ese apellido concreto ya que previamente lo inscribiste con un espacio en blanco al final de las letras.

El contenido de una variable literal

El símbolo dólar ($) cuando se coloca como **sufijo** de un nombre de variable hace que ese dato sea considerado como de **índole literal**, y por tanto el contenido es una mera **sarta de caracteres** cualesquiera, letras, cifras o signos. Una sencilla operación con estos literales puede ser la mostrada en las líneas 10 a 90 que te presentamos. Al usar el símbolo de sumar (+) con operandos que son datos literales, lo que se hace es **empalmar** (y puedes usar la palabra concatenar) los operandos formando una sarta de caracteres más larga, y asignar el resultado como valor de una nueva variable literal denominada de forma diferente.

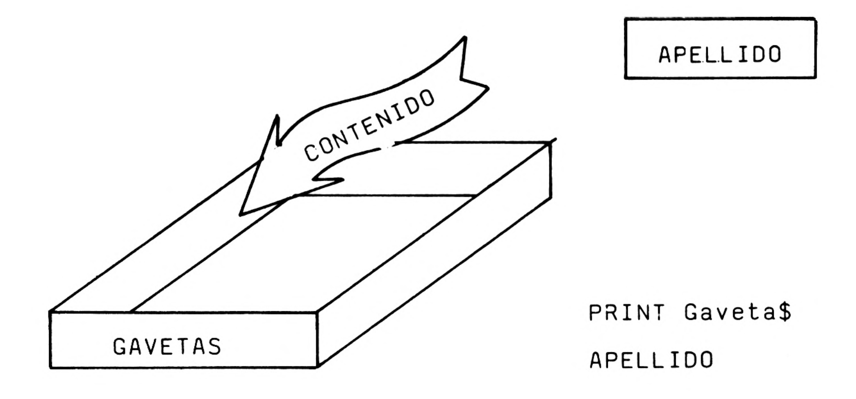


Figura 11.1: El contenido de una variable literal

```
10 INPUT "El nombre es A$ ";A$
20 INPUT "El apellido es B$ ";B$
30 LET z$=A$+b$
40 LET q$=a$+" "+b$
50 LET y$=a$+CHR$(32)+b$
60 PRINT
70 PRINT z$;"  Esto es A$+B$ "
80 PRINT
90 PRINT q$;"  Esto es A$+" + CHR$(34) +" "+ CHR$(34) + "+B$"
100 PRINT
110 PRINT y$;"  Esto es A$+CHR$(32)+B$ "
```

Las funciones de clave CHR$ y ASC

En las líneas 40 y 50 del programa anterior se ha logrado el mismo resultado por diferente método. El segundo método, el de la línea 50, es probablemente más elegante y profesional y un método más normal. El comando para que exponga el **carácter** cuyo código es 13, para lo que mandaríamos PRINT CHR$(13) tiene el mismo efecto que pulsar la tecla de **vale,** marcada **ENTER,** y no muestra ningún **carácter visivo** sino que da por acabada una línea pasando a la siguiente. Ese carácter no visivo es el conocido popularmente como **'retorno de carro'** (y en inglés carriage return) como herencia de las palancas en la máquina de escribir que hacían volver el carro a una nueva línea.

La función de clave CHR$ es una abreviatura de la palabra 'character': = símbolo, **carácter,** y se aplica sobre un número entero. Cada carácter tiene un **código** numérico que está normalizado por el convenio ASCII. Siempre que sea posible usar la función que proporciona el **carácter** a partir del **código** correspondiente a un determinado símbolo no visivo, es preferible usarlo así en lugar de usar el propio carácter encerrado entre paréntesis. La velocidad de tratamiento del ordenador se verá mejorada y el programa globalmente será más profesional. Desafortunadamente eso significa que para estudiar el listado del programa será necesario tener a mano una tabla de la codificación ASCII.

La función de clave ASC, abreviatura de ASCII tiene el efecto inverso, es decir entrega el **código** que corresponde a un carácter literal dado. Por ejemplo si mandamos que exponga el código que corresponde al carácter literal "E", tecleando PRINT ASC ("E") veremos que en pantalla nos muestra el número 69. El uso de esta función es un poco más especializado y restringido. También puede usarse como parte de las comparaciones que se usan en un bucle condicionado **mientras que...**, o una instrucción condicional **si... entonces...**

Contando el número de caracteres en un literal

La **longitud** de un dato literal, constante o variable, puede determinarse aplicando la **función** de clave **LEN** (abreviatura de LENGTH) y dándole como argumento el dato literal cuya longitud nos interesa:

```
10 INPUT "Dime un apellido ";b$
20 LET longi = LEN(b$)
30 PRINT "Ese apellido '";b$;"' tiene ";longi;" caracteres"
```

El uso inmediato de esta función cuando tratamos de ficheros es durante la etapa de generación y preparación. Como parte del procedimiento preparatorio el programa puede pedir al usuario que defina cada **campo** de información (cada casilla de la ficha) en lo referente al título o nombre del campo y a la máxima longitud del dato literal que va a ser reflejado en ese campo concreto de información. La ventaja de esta clase de estructura en el programa es que simplifica el diseño de las imágenes en pantalla que ayudan a inscribir los datos o a consultarlos cuando se está usando el fichero.

La función que entrega la **longitud** del literal puede usarse para constatar el número de caracteres de cada literal a medida que se va INponiendo lo tecleado a cada variable. Si es menor que la longitud prescrita quedará entonces almacenado como contenido de la variable múltiple con la que trabajamos. La estructura sería:

```
10 LET conta% = 1
20 INPUT "Cantidad de nombres ";Qanti%
30 INPUT "Maximo de caracteres ";Longi%
40 DIM nome$(qanti%)
50 WHILE conta% < Qanti% + 1
60   PRINT conta%;
70   INPUT "Nombre ";inpo$
80   IF LEN(inpo$) > longi% THEN GOTO 110
90   LET nome$(conta%) = inpo$
100  LET conta% = conta% + 1: GOTO 120
110  PRINT "Eso es muy largo "
120  WEND
```

Conversión a mayúsculas o minúsculas

Ambas **funciones** de nombres clave **UPPER$** (más arriba, caja alta) y **LOWER$** (más abajo, caja baja) son muy útiles porque generalmente un programador tiene preferencia por usar exclusivamente uno de los dos tipos de letras. La función que efectúan es convertir los caracteres de un dato literal, variable o constante, al tipo de letra correspondiente a la función empleada. Así, la de clave **UPPER$** convertirá todas las letras a **mayúsculas** y la función de clave **LOWER$** las convertirá a **minúsculas**. La ventaja esencial en cualquier programa es que sólo es necesario 'validar' la respuesta del usuario correspondiente a uno de los tipos de letra. El listado del programa también aparenta más profesionalidad. En relación con el usuario del programa no le importa en absoluto si ha pulsado la tecla de **enclavamiento a mayúsculas**, marcada CAPS LOCK, o no.

Formación de literales especiales

Para hacer un programa más eficaz, es una pequeña ventaja disponer y aprovechar todas las facilidades que el ordenador suministra. La función de clave **STRING$** (y ahí tenemos claramente el término **STRING:** = sarta, cordón, hilera, etc. y desde luego no es cadena: chain) es una de esas funciones útiles: con ella se puede formar un **literal** que conste de un número determinado de caracteres todos iguales a uno dado. Su formato es STRING$ (número de caracteres, "carácter").

```
10 LET T$ = STRING$ (5, "repite")
20 PRINT T$
RUN
```

Figura 11.6

Comprueba por ti mismo que este otro bucle reiterativo efectúa una tarea diferente:

```
10 FOR C = 1 TO 5
20 PRINT "repite";
30 NEXT
RUN
```

Figura 11.7

Un avance adicional pudiera ser el de sustituir los caracteres dados (como la constante literal dada como argumento en los ejemplos anteriores) por una **variable** literal o por el **carácter** que entrega la función de clave **CHR$** al ser aplicada a un **código**, como:

```
PRINT STRING$ (5, CHR$ (69))
```

Figura 11.8

La función de clave **SPACE$** es una forma especial de formar un **literal** que conste únicamente de espacios en **blanco** de una determinada longitud. Así por ejemplo puedes mandar que exponga en pantalla el dato contenido en SPACE$(5) con lo que habrás de arreglártelas para apreciar que han sido expuestos 5 espacios en blanco. Puedes usarla como forma alternativa de los **adverbios** que permiten TABular y separar datos con espacios en blanco, (de clave SPC) en un programa de tratamientos de texto por ejemplo.

A medida que tu habilidad como programador y tu bagaje de conocimientos se incremente, observarás que hay muchas más funciones disponibles como **utensilios programales** que te ayudarán con facilidad a que seas capaz de instruir al ordenador para que efectúe una tarea. Mi consejo es que lo hagas gradualmente y te vayas primero acostumbrando a las palabras claves fundamentales del BASIC. Cuando afrontas un problema aparentemente insoluble, debes consultar el **manual de referencia** del ordenador, y con un poco de imaginación observarás que la función que necesitas está allí para completar la tarea que deseas enseñar al ordenador. Cualquier tarea o función puede generalmente desglosarse en líneas de instrucciones de las más fundamentales. A medida que adquieras confianza 'hojea y ojea' el mencionado manual de referencia BASIC que se te regala con el ordenador para pertrecharte con esos utensilios de programación.

Capítulo Doce

Tratamiento de Literales con Mayor Aprovechamiento

La magia de cualquier programa de ordenador que funcione correctamente, es su capacidad para llevar acabo reiteradamente y con **fiabilidad** tareas repetitivas y normalmente aburridas una y otra vez sin cansarse nunca y sin dejar de ser exacto. Por ejemplo, busca una persona que tenga una dirección de 'castellana, 1' en un listín de 20.000 personas. Un programa bien diseñado acabaría dándonos la respuesta en breves segundo. Los procedimientos más comúnmente usados son los de búsqueda y **ordenamiento** de datos, y ambos ordenadores CPC 664/464 nos ofrecen diversas facilidades para buscar y ordenar. Esas facilidades son apreciadas eficazmente por el usuario del programa pero las opciones disponibles han tenido que ser incorporadas en el propio programa por el confeccionador del mismo. La esencia en la búsqueda de un nombre concreto, o en términos informáticos de un **literal** o **sarta de caracteres,** es usando la instrucción condicional:

SI a$=b$ ENTONCES HAGA HALLADO$="CIERTO"
ENDEMÁS HAGA HALLADO$="FALSO"

Esta instrucción para constatar si se ha hallado el valor buscado se incluye dentro de un bucle en que cada variable de una lista es sucesivamente cotejada con el valor de la variable buscada. Cuando las dos sartas de caracteres **concuerdan** (y en inglés verás lo de 'match') se da la búsqueda por acabada. Un ejemplo de este procedimiento es el que hemos empleado en el listín telefónico.

```
710 REM buscador de telefonos
730 CLS
740 PRINT "Debo buscar por ... "
750 PRINT: PRINT "<A> Nombre "
760 PRINT: PRINT "<B> Apellido "
770 WHILE K$<>"A" AND K$<>"B"
780   LET K$=INKEY$
790   LET K$=UPPER$(K$)
800   WEND
```

```
810 PRINT
815 LET nb$="???": LET ap$="???": LET tf=0
820 IF k$="A" THEN INPUT "INscribe el nombre";nb$
830 IF k$="B" THEN INPUT "INscribe el apellido";ap$
840 PRINT: PRINT "   "; nbr$,apl$,tfn$
850 WINDOW 1,40,10,21
860 LET contor = 0
870 WHILE contor<fichas
880    IF k$="B" THEN GOTO 900
890    IF nb$=nome$(contor) THEN LET tf=1:
         PRINT contor;nome$(contor),apel$(contor),tfno$(contor)
900    IF ap$=apel$(contor) THEN LET tf=1:
         PRINT contor;nome$(contor),apel$(contor),tfno$(contor)
910    LET contor = contor + 1
920    WEND
925 IF tf<>1 THEN LOCATE 10,8: PRINT "No me has INscrito esa persona "
930 GOSUB 1280 :REM a ver si SIGA
940 RETURN
```

La esencia de un **procedimiento** para **ordenar** datos (y en inglés verás lo de 'sort' que te recordará al sorteo pero no de lotería sino de los quintos cuando van a la mili) es probablemente un poco más complicado. Está centrado sobre la capacidad para decidir si una determinada sarta de caracteres es **menor que** o es **mayor que** otra sarta de caracteres concreta. En este caso estas **relaciones** se traducen en si están ordenadas alfabéticamente, es decir un literal es menor que otro cuando el primero está más cerca de la letra A que el segundo; y un literal es mayor que otro cuando el primero está más cerca de la letra Z que el segundo.

El método que el ordenador utiliza para reflejar cada **carácter** mediante un **código** ASCII numérico equivalente, facilita las cosas. Si el literal está formado por varios caracteres, se compararán individualmente cada carácter según la posición que ocupa. Por lo tanto resulta razonable decir que el código ASCII en el sistema de base 10 para la letra **A** es el 65 y para Z es 90. El problema es que para las letras en minúsculas los códigos ASCII en el sistema de base 10 van desde el 97 para la **a** hasta el 123 para **z.** Por lo tanto, si requieres ordenar literales que pueden ser de uno u otro tipo de letra, y prefieres que no tenga influencia en el ordenamiento, debes asegurarte que durante la rutina que efectúa la ordenación usas siempre las funciones para convertir a **mayúsculas** o a **minúsculas**, según prefieras, que están disponibles en tu CPC 664/464.

Un método poco elegante pero que funciona para ordenar tres variables literales según orden alfabético ascendente es el mostrado a continuación. No hemos incluido una validación para aceptar tanto mayúsculas como minúsculas, de manera que claramente no está hecho a 'prueba de tontos'.

Sin embargo sirve como útil demostración del procedimiento necesario para exponer una lista cuyos elementos están en orden alfabético.

```
100 LET CONTa% = 1
110 PRINT "Dime los 3 LITERALes a ordenar"
120 PRINT "Teclealos separados por comas"
130 INPUT ;a$,b$,c$
140 PRINT:PRINT
150 PRINT "Puestos en orden alfabetico, son:"
160 PRINT
170 WHILE conta% < 4
180    IF a$<=b$ AND a$<=c$ THEN LET buz$=a$: tf%=1
190    IF b$<=c$ AND b$<=a$ THEN LET buz$=b$: tf%=2
200    IF c$<=a$ AND c$<=b$ THEN LET buz$=c$: tf%=3
210    PRINT buz$
220    IF tf%=1 THEN LET a$="zzzzzzzzzz"
230    IF tf%=2 THEN LET b$="zzzzzzzzzz"
240    IF tf%=3 THEN LET c$="zzzzzzzzzz"
250    LET conta% = conta% + 1
260    WEND
270 PRINT
```

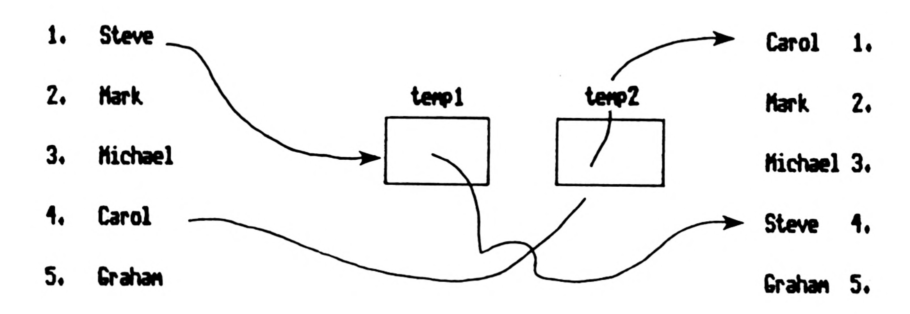
La teoría subyacente es la de encontrar el literal que está más cerca de la letra A; ese literal concreto es luego destruido, o más precisamente cambiado para que esté formado por una serie de z, que alfabéticamente será el último de la lista ordenada. De esta manera se puede encontrar el siguiente en orden alfabético, exponerlo en pantalla, y luego después alterarlo para que sea otra serie de z. El tercero en orden alfabético se encuentra a continuación, se expone, se altera a z, y así sucesivamente.

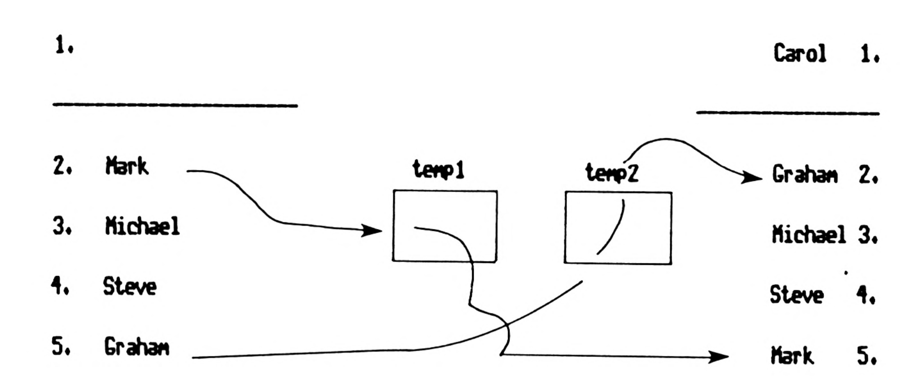
El problema de este método es que lleva bastante esfuerzo y tiempo la tarea de escribirlo en el programa cada vez que una sencilla combinación para más tres literales ha de ser ordenada.

Se han inventado muchos métodos diferentes de ordenamientos de datos literales. Cada uno tiene su propia ventaja: algunos tienen en cuenta mayúsculas y minúsculas, otros destruyen el contenido de la variable original de manera que no pueda usarse otra vez, y otros no lo hacen; algunos son mucho más rápidos en efectuar la tarea que otros. Todos usan una serie de canjes o intercambios del contenido de las variables de unas a otras.

La teoría es bastante directa: cada elemento de la lista es examinado repetidamente, usando el primer elemento como **normal** hasta que se encuentra otro en la lista que está más cerca alfabéticamente de la letra A; ese item se convierte entonces en el **estándar** contra el cual cotejar los otros. El proceso se continúa hasta que se ha alcanzado el final de la lista. El estándar corriente en ese momento es el que se expone en pantalla. Se desprecia ese elemento estándar y se convierte a un literal alfabético formado por una serie de z. Se vuelve a tomar como estándar en esta segunda pasada el primer elemento y se repite el proceso hasta encontrar el más pequeño de los que quedan. El proceso continúa sacando uno a uno los elementos por orden alfabético hasta que todos hayan sido ordenados y expuestos en la pantalla.

Hay otros métodos que usan tablas monodimensionales y realmente alteran la posición que el elemento ocupa en la **ristra,** cambiando precisamente el **subíndice** ordinal del elemento dentro de la variable múltiple. El contenido de las dos variables es almacenado temporalmente en otras variables y luego son **canjeados** de acuerdo con el orden correcto. El número de elementos comparados se ajusta de manera que todos aquellos que ya están colocados en su sitio no vuelven a examinarse. La Fig. 12.1 ilustra la procesión de eventos que tiene lugar. El diagrama estructurado pudiera ser algo como el de la Fig. 12.1, demostrando otra aplicación muy útil para generar diagramas estructurales de una tarea específica. Mira a ver si puedes generar tus propios métodos de búsqueda y archivarlos en tu cinta o disco. Comienza a partir del punto de meramente pergeñar las ideas en la forma de un esbozo o esquema de diagrama.





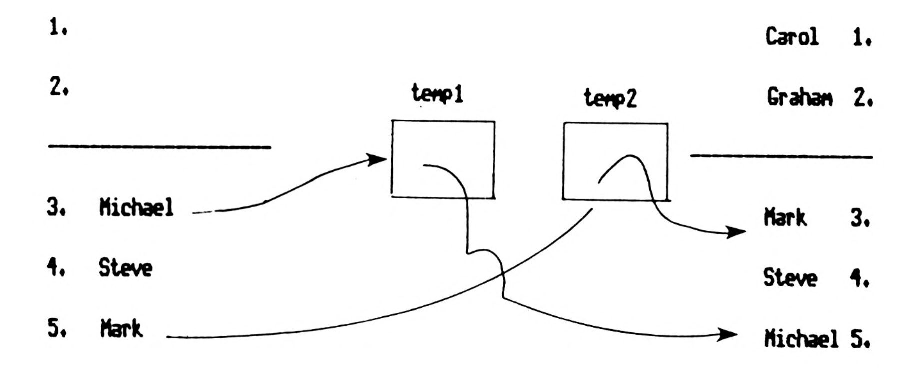


Figura 12.1: Ordenamiento alfabético - el uso de depósitos temporales

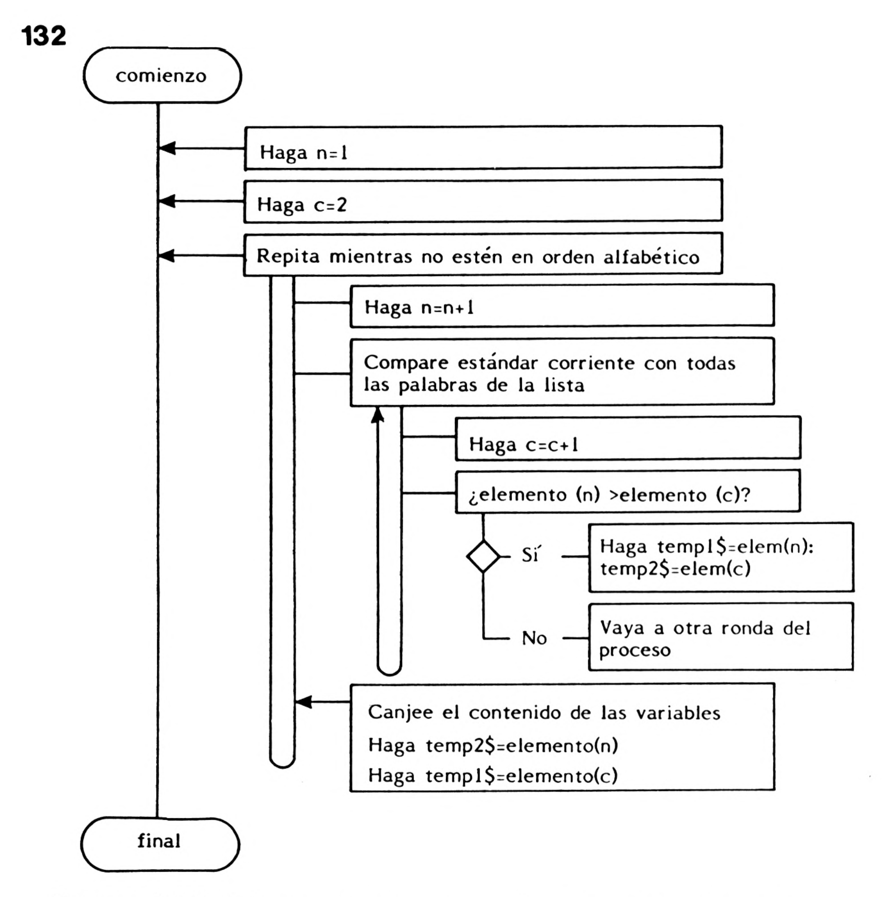


Figura 12.2: Diagrama estructural mostrando el ordenamiento hecho

Segregamiento de subliterales

Hay varias **funciones** disponibles que pueden introducirse en la estructura de un programa para dotarlo de una mayor flexibilidad al determinar la parte exacta o la sección de un literal que es la que deseamos buscar u ordenar. Las funciones que permiten **segregar** una determinada sarta de caracteres de otra sarta de caracteres dada, son las que tienen por clave LEFT$ (izquierda, los **an**teriores) RIGH$ (derecha, los **ul**teriores) y MID$ (abreviatura de MIDDLE: = **medianeros**, o del medio).

La sintaxis de cada una es la que aparece en la Fig. 12.3, y debe tenerse en cuenta que cada una usa un dato literal del que segregar el subliteral, y un parámetro determinado para indicar la longitud del subliteral, y que todos entregan como resultado un dato literal cuyos caracteres son los que se extrajeron del argumento literal dado.

Este subliteral así formado puede tomar los caracteres comenzando por los primeros o **anteriores**, los de la parte derecha o **ulteriores**, o los **medianeros**, exigiendo en este caso un parámetro adicional para fijar la posición a partir de la cual se extraen los caracteres. La función de clave **LEFT$** segrega los caracteres primeros del literal dado, **MID$** segrega un subliteral tomando los caracteres intermedios y **RIGHT$** los colocados hacia el final del literal dado. Cuántos caracteres forman el subliteral segregado queda determinado por los parámetros de la función.

```
PRINT left$("computadora",3)

com

PRINT mid$("comparadora",4,4)

para

PRINT right$("computadora",4)

dora
```

Figura 12.3: Segregamiento de subliterales

Para demostrar los fundamentos de estas funciones considera las siguientes líneas de programa; tecléalas en tu ordenador como un programa y mándale que lo **rule:**

```
10 INPUT "Dime una palabra de 3 silabas "; liter$
20 INPUT "Posicion donde empieza la silaba 2 ";comi2%
30 INPUT "Cantidad de letras en esa silaba ";lon2%
40 LET med$ = MID$(liter$,comi2%,lon2%)
50 PRINT "La segunda silaba de esa palabra es ";med$
```

Pertenencia de un subliteral a un literal

La función de clave **INSTR**, es de diferente naturaleza, ya que entrega como resultado un **número** entero que representa la posición **ordinal** a partir de las cuales **coinciden** dos literales dados. Se dice normalmente que indica si un subliteral **pertenece** a un literal determinado. Su formato es el mostrado en al Fig. 12.4 y puede usarse en las maneras mostradas en los programas de las Fig. 12.5 y 12.6.

INSTR (Comienzo, literal, subliteral)

Figura 12.4: La función de PERtenencia para literales

```
110 CLS
120 DATA Ramon,Columela 22
130 DATA Rosa,Capitan Haya 77
140 DATA Javier,Serrano 123
150 DATA Isabel,Avenida del Generalisimo 234
160 LET d=4
170 PRINT "Posicion de la palabra 'Haya'": PRINT
180 FOR c=1 TO d
190   READ nombre$(c),calle$(c)
200   PRINT INSTR(5,calle$(c),"Haya");
210   PRINT TAB(10) calle$(c)
220   NEXT
230 PRINT
```

Figura 12.5: Uso de la función INSTR

```
100 CLS
110 DATA Ramon,Columela 22
120 DATA Rosa,Capitan Haya 77
130 DATA Javier,Serrano 123
140 DATA Isabel,Avenida del Generalisimo 234
150 INPUT "Posicion de comienzo de la busqueda";comi
160 INPUT "El nombre, el numero o el tipo de calle";abuscar$
170 LET d=4
180 FOR c=1 TO d
190   READ nombre$(c),calle$(c)
200   NEXT
210 FOR c=1 TO d
220   LET buscado=INSTR(comi,calle$(c),abuscar$)
230   IF buscado>0 THEN LET testigo=1: GOSUB 270
240   NEXT
250 IF testigo=1 THEN GOSUB 320
260 END
```

```
270 REM rutina de busqueda
280 LET a=a+1
290 LET buscado$(a)=nombre$(c)
300 LET buscado1$(a)=calle$(c)
310 RETURN
320 REM imprime criterios
330 CLS
340 PRINT abuscar$: PRINT
350 FOR b=1 TO a
360   PRINT buscado$(b);CHR$(32);buscado1$(b)
370   NEXT
380 RETURN
```

Figura **12.6:** Más sobre el uso de la función INSTR

Ahora experimenta poniendo en práctica tus ideas y sacando tus conclusiones. Recuerda que si el subliteral no pertenece al literal dado, el resultado entregado por la función de **pertenencia** es el número 0; que como sabes se considera como 'falso'.

Capítulo Trece

Instrucciones para el Aporte de Datos

Las palabras del BASIC de claves READ (leer) y DATA (datos) funcionan conjuntamente en todas las circunstancias, para aportar datos a las variables y parámetros utilizadas en un programa. El uso del comando DATA es un método de hacer que el ordenador **cense** una lista de datos como parte inherente del programa que podrán ser **tomados** sucesivamente como datos de variables. Su utilización primordial es para reflejar información que permanecerá siempre igual cada vez que el programa se utilice. El comando de clave READ hará que el ordenador **tome** (efectúe la lectura) del dato correspondiente del **censo** y asigne el dato tomado como valor de la variable mencionada en el comando de clave READ, tal y como se indica en las Fig. 13.1 y 13.3.

```
50 DATA 1,12,13,14,14,19,18,17,16,15
60 READ numero
65 MODE numero: PEN numero+2
70 FOR contador=1 TO 9
80   READ posicion
85   PRINT posicion
90   PRINT TAB(posicion) "Imagen en Pantalla"
100  NEXT
```

Figura 13.1: Censo de datos tomados como valores de variables (i)

```
100 DIM letra$(20)
110 DATA Capa,asa,sana,anida," ",&
120 DATA " ",Cala,asma,maza,alma
130 WHILE letra$(conta)<>"alma"
140   LET conta=conta+1
150   READ letra$(conta)
160   WEND
170 FOR c=0 TO conta
180 PRINT LEFT$(letra$(c),1);
190 NEXT
```

Figura 13.2: Censo de datos tomados como valores de variables (ii)

```
100 READ numero
110 CLS
120 PRINT "Num. de CTA.","Apellido":PRINT
130 FOR c=1 TO numero
140 READ numero,nombre$
150 PRINT numero,nombre$
160 NEXT
170 PRINT
180 DATA 4
190 DATA 1423,Sierra
200 DATA 6743,Arroyo
210 DATA 2341,Lejido
220 DATA 7453,Porras
```

Figura 13.3: Censo de datos tomados como valores de variables (iii)

Si esbozamos el diagrama de estos breves programas, es posible que comprendamos mejor la naturaleza y utilización de estos comandos para aporte de datos. Examina los diagramas de las Fig. 13.4 a 13.6 y observa la ausencia de mención explícita de las instrucciones reales de clave DATA. Hay dos razones: primeramente no son realmente comandos en el sentido de 'verbos' que mandan realizar alguna acción al ordenador, si no que meramente él las incluye en un 'censo' considerándolas una detrás de otra aunque estén colocadas en diversos puntos del programa; en segundo lugar, si el diagrama estructurado se escribiera pormenorizadamente la necesidad de los **datos** estaría completamente documentada en el mismo momento en que son **tomados** como valores de variables.

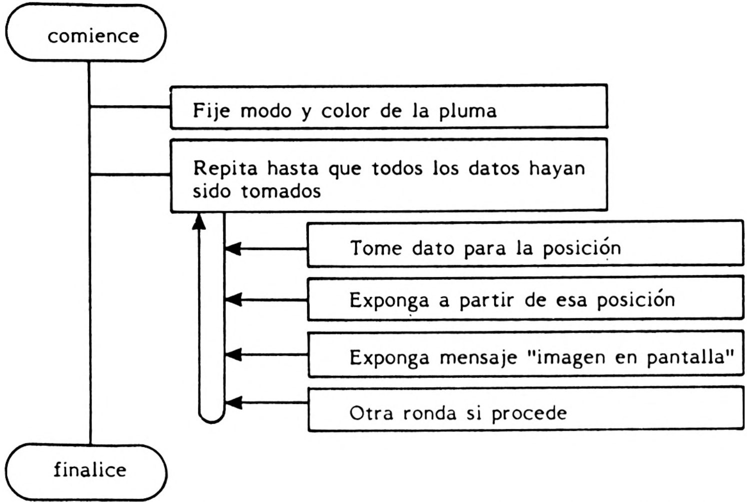


Figura 13.4: Diagrama del programa en la Fig. 13.1

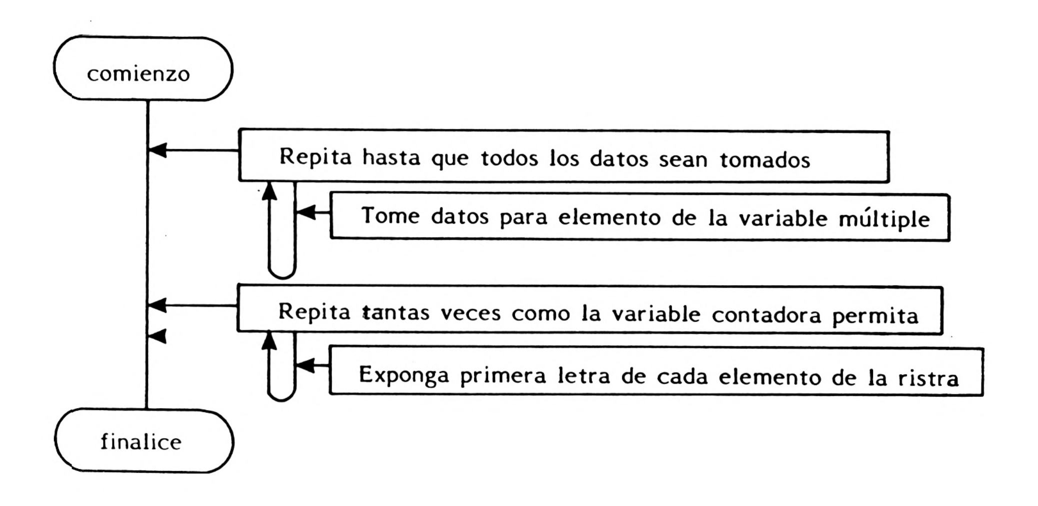


Figura 13.5: Diagrama del programa en la Fig. 13.2

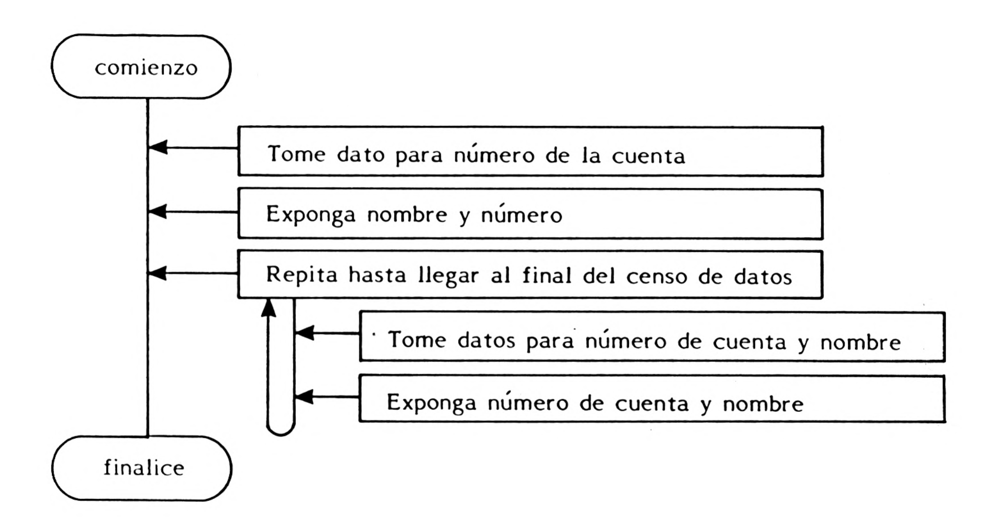


Figura 13.6: Diagrama del programa en la Fig. 13.3

El uso de estas instrucciones es bastante directo y sigue las reglas:

1) Cada pieza de información en el censo de datos está separada de la siguiente por una coma (,).

2) Se pueden incluir datos numerales y literales en cualquiera de las instrucciones DATA censadas en el programa.

3) Cuando se **toma** un **dato** del censo para darlo como valor de una variable, el dato y la variable han de concordar exactamente, numeral o literal ambos.

4) La primera instrucción de clave **READ** hará que se **tome** el primer dato de la primera instrucción de clave **DATA** que haya reflejada en el programa. La segunda toma de datos hará que a la variable se le asigne el segundo dato del censo total, y así sucesivamente. (El propio ordenador lleva un **puntero** internamente para ir marcando cual es el dato del censo que le corresponde tomar en cada momento).

5) Si requieres que el puntero interno **repunte** a una instrucción específica del programa, puedes usar el comando de clave **RESTORE** (restaurar las condiciones iniciales) seguido del número de línea deseado.

Uno de los usos más prácticos de esta posibilidad de **tomar datos** es la de repetir una subrutina concreta en diversas ocasiones dentro de un programa, pero con diferentes **parámetros** cada vez que se recurre a ella para obtener una acción de matiz diferente. Así, puede que haya cinco diferentes variables dentro de esa subrutina, pero en cada una de las quince ocasiones en que se apela a la subrutina se toma para cada variable un dato diferente. Es una técnica relativamente sencilla de programación incluir las quince combinaciones de valores de las variables en quince instrucciones de clave DATA. Al comienzo de la subrutina una instrucción de clave **READ** recaba los nuevos datos necesarios para esa ocasión y los coloca como contenido de las variables involucradas.

Las instrucciones de **tome... datos...** son un instrumento programal que puede permitir volar la imaginación. Comienza experimentando con algunos gráficos sencillos, usando variables y **datos constantes** para formar el censo de donde **tomar** los parámetros necesarios para variar ligeramente las formas o los colores. Inténtalo.

Capítulo Catorce

Ampliación de Variables-Tablas

La forma más sencilla de imaginar el concepto de datos **variables** es el mostrado en la Fig. 14.: la variable se representa por una gaveta o cajetín rotulado con un **nombre** distintivo y cuyo contenido es el **valor** asignado en la variable en cada momento.

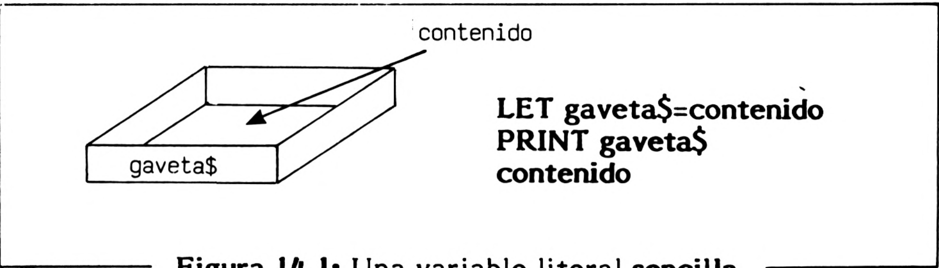


Figura 14.1: Una variable literal **sencilla**

Con el fin de producir un listín telefónico y otros diversos programas de ejemplo, ha sido imprescindible utilizar una forma diferente de variable. Esta variable es de la clase **múltiple**, dado que en realidad es un colectivo de variables que forma una **ristra** ordenada (en inglés oirás 'array': = ringla, conjunto arreglado, escuadrón, etc.). Lo importante con estas variables múltiples, o **tablas**, es que poseen todas un mismo nombre para indicar al colectivo estando cada una de los elementos identificados mediante un **subíndice** o **subscrito** que se coloca entre paréntesis como sufijo del nombre colectivo; por ejemplo nome$(2) señalará a la variable 2 del colectivo nome$. Cada uno de estos elementos de la tabla, aunque tengan el mismo nombre colectivo, puede contener diferente valor, ya que el **nombre** se refiere a la **ristra** de cajetines y cada cajetín está debidamente señalizado por un número correlativo. Al comienzo de cada programa, la cantidad de elementos que van a componer la tabla ha de establecerse para que el ordenador **ocupe** el espacio que corresponde a las **DIMensiones** de la ristra de cajetines. Se efectúa mediante la instrucción de clave **DIM** seguida del nombre colectivo y entre paréntesis el número de elementos que componen la tabla o variable múltiple (en realidad la ristra de cajetines tiene uno más -el numerado como 0- de los que figuran en la instrucción de clave DIM).

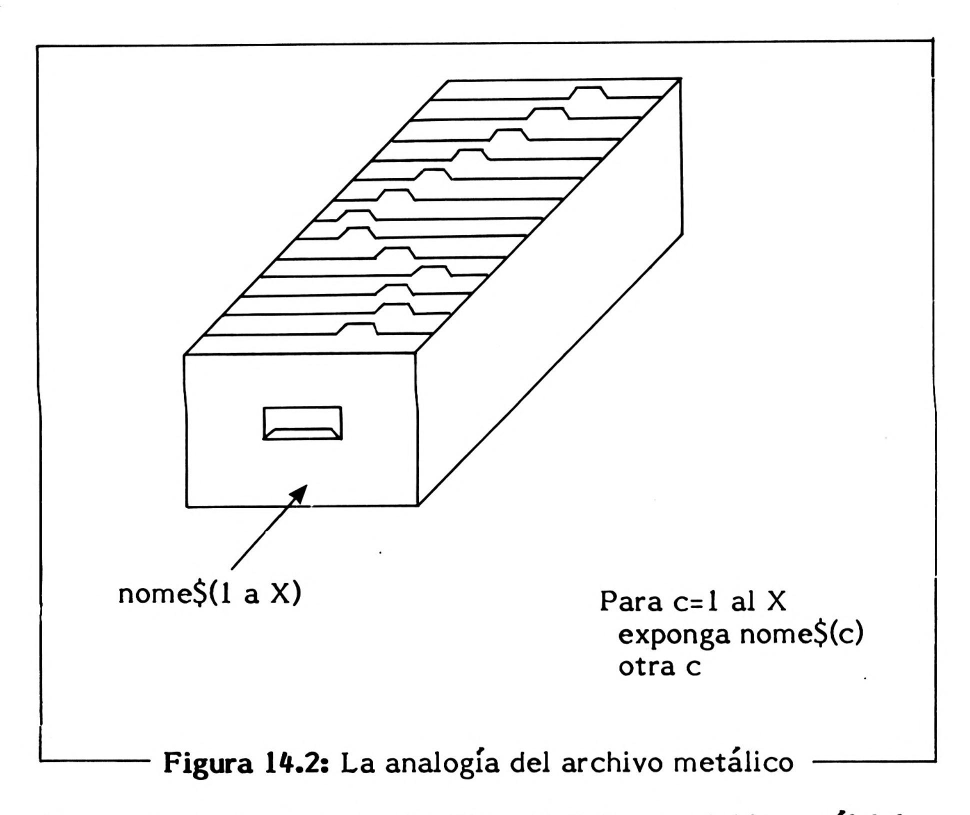


Figura 14.2: La analogía del archivo metálico

Como se demuestra en la Fig. 14.2 las variables **múltiples** pueden verse mucho más como un cajón con sus gavetas separadoras perteneciente a un fichero o archivo metálico. La etiqueta en la parte frontal del cajón establece el nombre de los elementos que hay dentro del cajón y la cantidad de gavetas en que está divido. Cada una de las gavetas del cajón está a su vez **numerada** correlativamente y dentro de ellas tienen un cierto contenido. El cajón contiene pues toda la ristra de variables pertenecientes a esa tabla. La belleza de esta analogía es que al igual que ocurre con un fichero o agenda personal, cada cajón puede contener diversas variables que guardan una estrecha relación entre sí al corresponderles siempre el mismo **número de referencia** para las que pertenecen a un mismo titular, por ejemplo, el nombre, la fecha de nacimiento, la dirección, etc., pueden ser diferentes variables múltiples de manera que los elementos que tengan el mismo número de referencia correspondan a una misma persona.

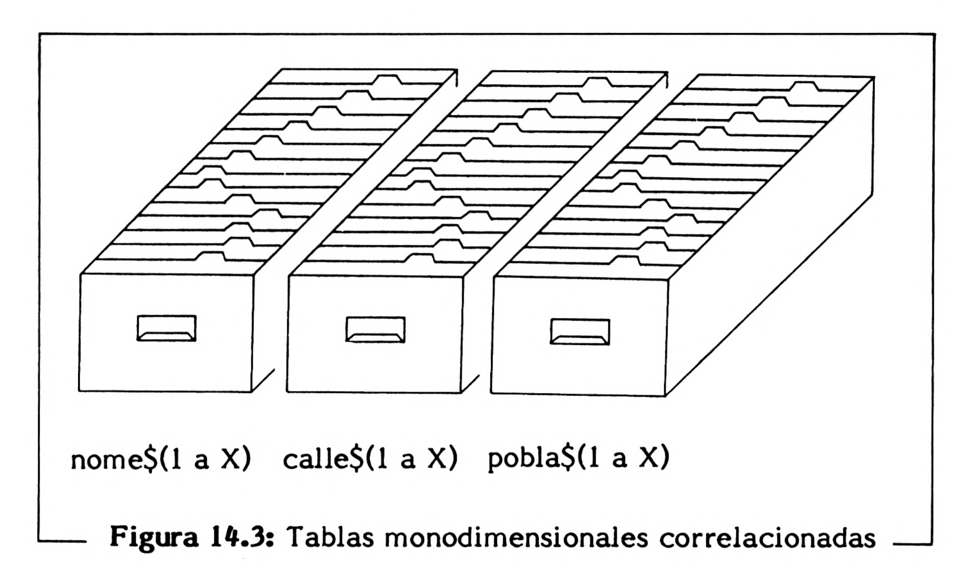


Figura 14.3: Tablas monodimensionales correlacionadas

Hasta ahora sólo hemos considerado tablas o variables múltiples de una sola dimensión (de ahí que hayamos usado el nombre de **ristras** que realmente le conviene). Considera ahora la Fig. 14.4 donde cada pieza de información involucrada en cada fichero personal está contenida en una variable diferente con dos subíndices, en que por contraposición tendremos una **ringla** o **tabla cuadrangular** de gavetas. Así la Fig. 14.4 demuestra la forma de establecer una tabla con 100 elementos y en cada uno 7 campos o piezas de información. (El equivalente sería definir 7 tablas monodimensionales de nombre diferente y de 100 elementos cada una, manteniendo una estrecha relación mediante el **número de referencia** de cada una).

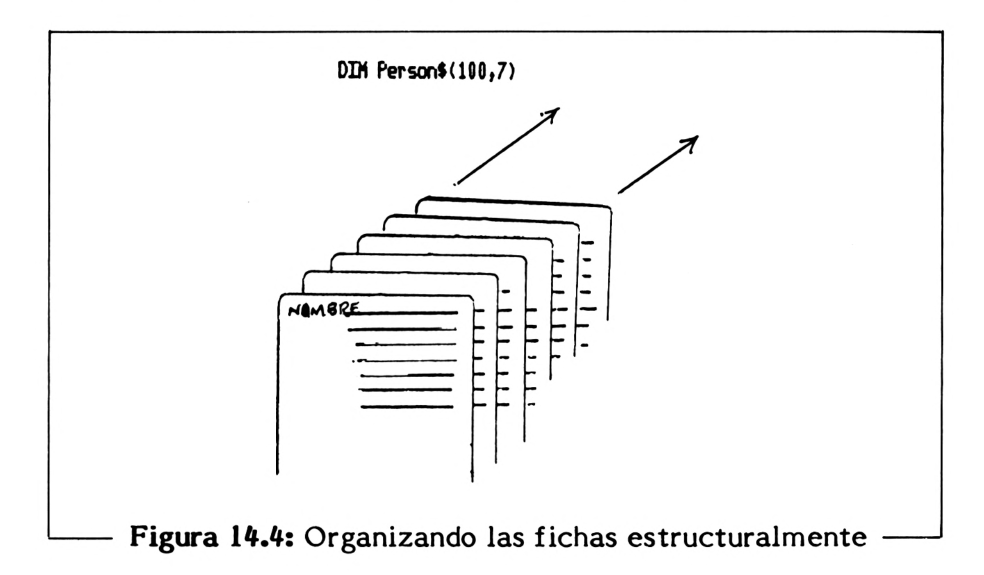


Figura 14.4: Organizando las fichas estructuralmente

Esto podemos avanzarlo un paso más, como la Fig. 14.5, donde 100 ficheros personales pueden contener 7 áreas diferentes de información, y cada área teniendo 3 campos de información. Es una manera elegante -aunque al principio no lo parezca- de incluir en un programa **colectivos arreglados** de datos de donde obtener información exactamente, de manera **directa,** y por tanto eficaz. Pero no te dejes llevar muy lejos: el problema con esta técnica concreta de programación es que puede usar de forma extravagante y desperdiciadora la memoria del ordenador.

Las variables múltiples, monodimensionales o polidimensionales, ristra o ringlas, pueden ser tanto numerales como literales según la naturaleza de sus datos. No te confundas. No des valores numerales a una variable múltiple literal y esperes que efectúe operaciones aritméticas con esos valores. Puedes efectuar operaciones literales pero no aritméticas y viceversa.

La estructura de las líneas de programa para **recorrer** toda una tabla:

```
10 DIM celdilla(3,3)
20 PRINT
25 FOR r=1 TO 3
30 FOR c=1 TO 3
40 PRINT celdilla(c,r);
50 NEXT
60 NEXT
```

El uso de variables múltiples dentro de un bucle hace excepcionalmente fácil traer de un disco o de una cinta todo un **fichero de datos** y asignar esos datos 'arregladamente' a los elementos de la tabla, exponer el contenido en pantalla, comparar los contenidos de cada elementos de la tabla con un estándar, volver a depositar dichos contenidos una vez modificados como fichero de datos en cinta o en disco, etc.

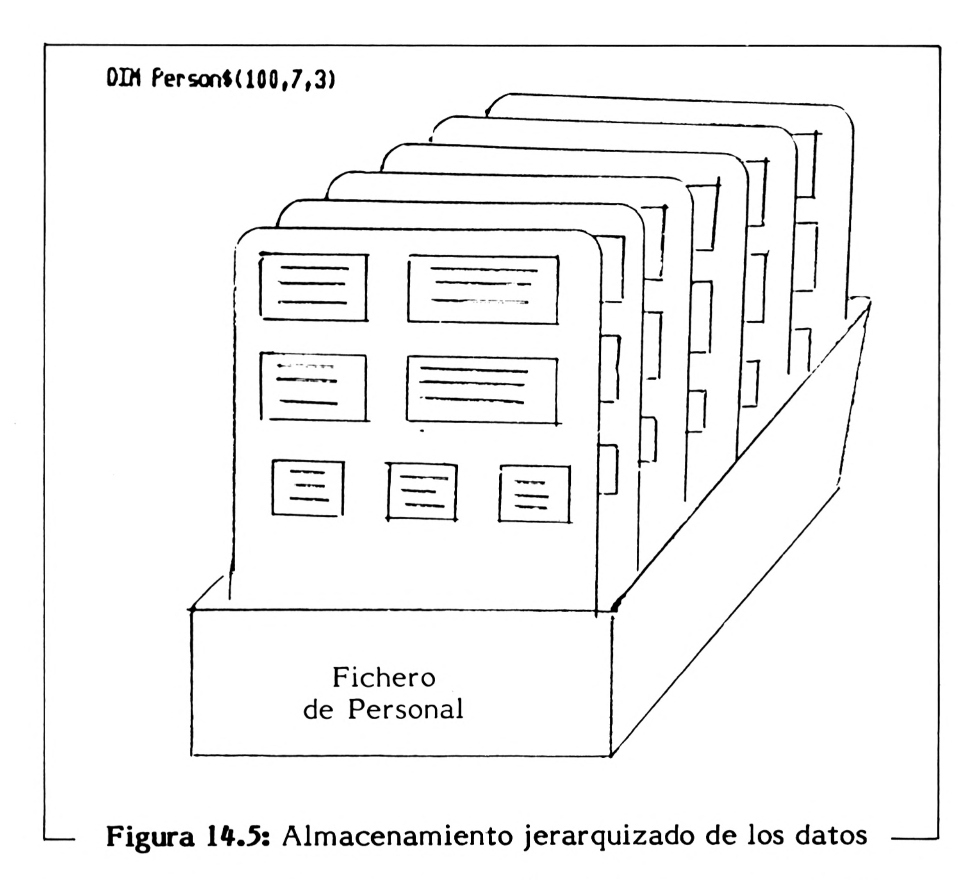


Figura 14.5: Almacenamiento jerarquizado de los datos

El uso de estas variables múltiples es muy amplio y probablemente sólo esté limitado por la propia imaginación del programador en su aplicación.

Capítulo Quince

Examen del Fichero, Cartela para IN/EXposición de Datos Contenidos

El punto crucial al confeccionar un programa para bases de datos que pueda tratar grandes cantidades de información es el método por el que el usuario ingresa los datos y la información necesaria para generar el fichero. El método real de inscribir los datos estará determinado por las instrucciones del programa, es decir: si los títulos están normalizados y prescritos o van a ser determinados por el usuario del programa en el momento de preparar ese fichero en concreto. Esta tarea es la se conoce como 'definición del fichero'. Mediante un programa que emplee esta técnica, un sencillo programa para bases de datos puede tener numerosas aplicaciones diferentes.

Una rutina para generación de un fichero que permita definir la estructura de las fichas -el **formato**- necesita pedir los siguientes datos al usuario:

1) Número de los campos de la ficha.

2) El título o nombre identificativo del primer campo.

3). El título o nombre identificativo del segundo campo.

...y así sucesivamente hasta que el número de campos establecido en la primera petición haya recibido su título.

Cada respuesta del usuario quedará archivada como contenido de una variable y grabada en cinta o en disco para ser usada durante el proceso de establecimiento de las condiciones iniciales para una aplicación concreta.

Los programa profesionales usados en el mundo industrial y comercial disponen de esta clase de facilidades como norma estándar. Durante la preparación de la base de datos, el usuario puede definir diversas opciones, establecer los rasgos característicos de cada aspecto del fichero, desde el número total de campos en una ficha hasta los campos que aparecen en pantalla en cualquier momento dado. Todas estas técnicas profesionales para la creación de ficheros pueden conseguirse mediante la programación en BASIC siempre y cuando estén razonadamente planificadas de una manera estructural.

El procedimiento podría típicamente ser:

1) Decidir exactamente las facilidades disponibles al usuario con ese programa para base de datos.

2) Decidir exactamente aquellos aspectos que el usuario tenga permitido determinar.

3) Crear una estructura que archivará las respuestas del usuario como un fichero de datos **paramétricos.**

4) Trazar la estructura del programa para la base de datos por medio de diagramas estructurales.

5) Escribir el programa basándose en 'los principios al principio'; y así cuando se definan los valores constantes del programa, con los dados por el usuario se sustituirán los asignados a las variables. Cuando el programa se utilice, esos datos serán traidos inicialmente desde el fichero de datos durante la etapa de preparación.

Términos técnicos usados a menudo

El uso de **cartelas,** también llamadas plantillas o moldes para INgreso y EXposición de datos, se usan en los programas profesionales para preparar las imágenes en pantalla concernientes a la información de la base de datos. Esencialmente, el usuario del programa controla la posición del cursor de manera que así determina la posición de la información expuesta en pantalla. Cada **campo** de información puede individualmente ubicarse en un sitio de la pantalla, situando el cursor mediante las teclas de desplazamiento marcadas con flecha ascendente, descendente, a derecha y a izquierda. En lo que se refiere a la programación en BASIC, la técnica para producir esta clase de programas envuelve la constatación de la posición ocupada por el cursor según cada eje de la pantalla, colocando esa información como valor de una variable preparada para ello. Cada campo de la ficha, tendrá pues una pareja de coordenadas que definen su posición en la pantalla, que será luego aprovechada al exponer los datos, mediante el comando para que **ubique** el cursor, de clave **LOCATE**-

Una **carátula** en pantalla es un término conceptual, que describe los campos de información que aparecen en la pantalla, junto con los recuadros o enmarques correspondiente. Imagina un **impreso** en blanco sobre la pantalla: la información reseñada por ejemplo en cada ficha personal se mostraría a través de cada carátula. Planificando cuidadosamente el programa para bases de datos, se pueden preparar diferentes carátulas de pantalla con el fin de exponer diferentes aspectos de cada ficha en concreto. Una **ficha** (y en inglés verás 'record', de recordar, registrar en un documento) es una colección de **campos** que guardan relación entre sí; un ejemplo puede ser la información contenida sobre un individuo en un fichero de personal.

Un **fichero**, que en inglés verás 'file', para indicar que es un 'filón', o serie 'enfilada' de elementos de una naturaleza **homogénea.**

Un **campo**, que en inglés verás el término 'field', es un trozo único de información dentro de una ficha o registro, y es el equivalente lo que llamamos normalmente **casilla** en un impreso formal, y un ejemplo puede ser la fecha de nacimiento o el nombre de la calle.

Capítulo Dieciséis

Tratamiento de Variables Numerales-Actualización Automática de Fichas

Las aplicaciones de bases de datos, tales como el listín telefónico, conciernen primordialmente a la manipulación de campos **literales** de información. Con el fin de presentar una panorámica completa de la programación vamos ahora a considerar la posibilidad de usar campos numerales dentro de un programa para bases de datos. Todos los campos numéricos pueden desde luego sufrir operaciones aritméticas sobre sus contenidos o valores. Imaginemos expuesta en pantalla la información contenida dentro de una base de datos que guarda relación con lo ganado mensualmente por un individuo a lo largo de un período anual, tal como en la Fig. 16.1. El usuario del programa únicamente necesita inscribir los descuentos por impuestos y reflejar los 12 datos de la cantidad pagada cada mes. El programa calculará luego todos los otros datos pertinentes, los reflejará como contenidos de las variables apropiadas y subsiguientemente expondrá los resultados en pantalla. El programa tal y como se muestra en la Fig. 16.2 es meramente el esquema de una base de datos mayor. Claramente, el programa requerirá la facilidad de tratar el fichero para guardar los datos inscritos por el usuario, sobre disco o cinta. En su forma actual los 12 pagos mensuales tienen que inscribirse uno trás otro sin apagar el ordenador.

Examinemos el listado del programa cuidadosamente. Ensáyalo tecleando en el ordenador y mandándole que lo **rule.** Sigue las instrucciones y observa como calcula el contenido de los otros campos. Por complitud, sigue el programa mientras analizas el diagrama estructural mostrado en la Fig. 16.3.

El uso de las funciones y operaciones aritméticas es directo. Los nombres de las variables numerales intervienen en una fórmula. El contenido de cada variable es sustituido por su valor al evaluar la fórmula, y las operaciones aritméticas se efectúan de acuerdo con las reglas de **precedencia,** de **prioridad:**

Pagos de este mes: __________

Enero ______
Febrero ______
Marzo ______
Abril ______
Mayo ______
Junio ______
Julio ______
Agosto ______
Septiembre ______
Octúbre ______
Noviembre ______
Diciembre ______

Salario Bruto ______
S. Social ______
Retenciones ______
Impuestos ______
Salario Neto ______

Figura 16.1: Imagen en pantalla de una ficha para ganancias personales

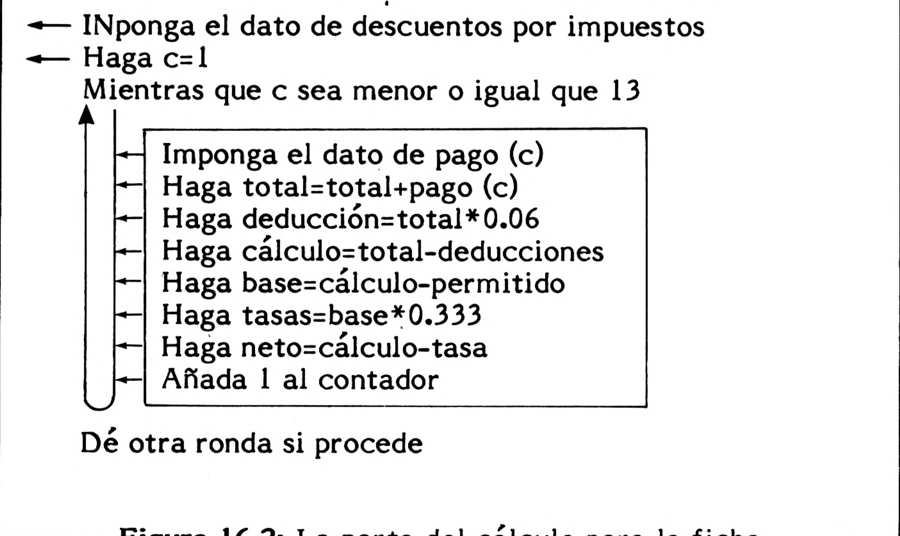


Figura 16.2: La parte del cálculo para la ficha de ganancias personales

1) La aritmética se efectúa desde la izquierda hacia la derecha.

2) Cada elemento encerrado entre paréntesis tiene **prioridad** en las operaciones, i.e. se evalúa en primer lugar.

3) Se calculan todas las multiplicaciones

4) Se calculan todas las divisiones.

5) Se calculan todas la sumas.

6) Se calculan todas las restas.

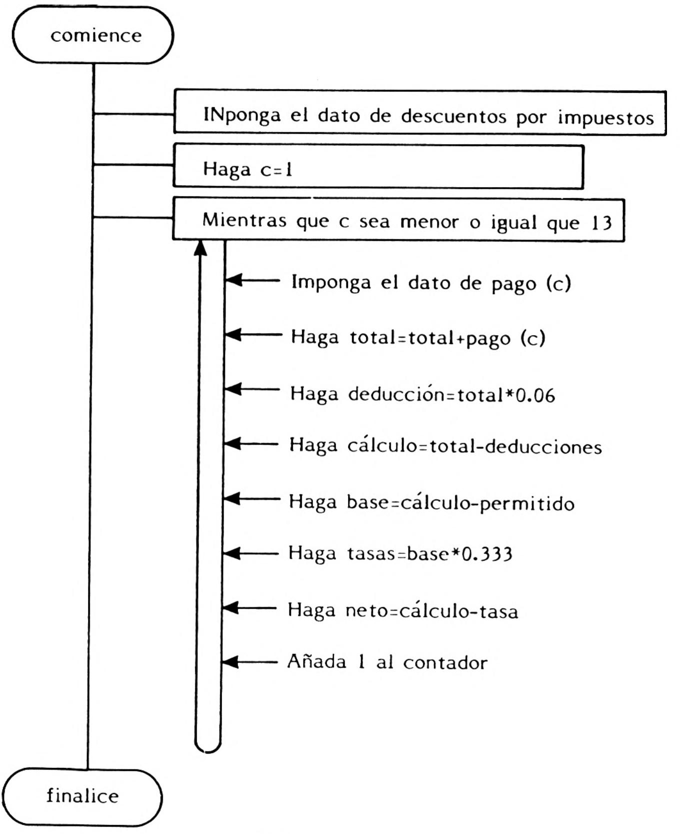


Figura 16.3

La **fórmula** adopta la sintaxis correspondiente a una instrucción de **haga... igual...** con la clave **LET** que es opcional. La variable a la que se 'asigna' como contenido el resultado obtenido al computar la fórmula es la que está colocada a la izquierda del signo igual:

Haga paga=tarifa por número de horas
Haga población del colegio=número de alumnos + número de profesores

---

Además de efectuar las cuatro operaciones básicas de suma, resta, multiplicación y división hay funciones y operaciones adicionales. Estas funciones incluyen diversos métodos para **redondeo** de números, conversión de números negativos a positivos, y cálculos de diversas funciones trigonométricas ya sean en **grados** o en **radianes**. Mediante el **adverbio** USING para la instrucción de exposición de datos se puede elegir la forma en que se muestran los datos numerales. Todas estas facilidades pueden incorporarse fácilmente en una rutina para cómputo numérico dentro de un programa. Consulta el **manual del usuario** del ordenador para estudiar los detalles, el excelente **prontuario** del Amstrad.

Incorporando una instrucción de asignación, tal como haga **c=c+1** dentro de un bucle condicionado, conseguimos una sencilla rutina para ir incrementando el contador en cada ronda. Eso puede ampliarse un escalón más haciendo que **haga c=c+a,** por lo que el nuevo valor del contador es igual al valor previo más una cantidad reflejada por 'a' que puede ser el valor INpuesto por el usuario cada vez que se lleve a cabo ese bucle. El contenido inicial de la variable contadora puede fijarse a 0. El resultado es una rutina acumuladora que irá agregando en dicha variable 'c' los valores ingresados por el usuario. La Fig. 16.4 demuestra ampliamente este arreglo:

```
FOR b=1 TO 10
LET A=A+B
NEXT

            o bien

C=20
FOR B=1 TO 10
LET Total=Total+C
LET C=C+1
NEXT        lo mismo que 20+21+22+23+24+25+26+27+28+29+30
```

Figura 16.4: Muestra de una secuencia o procesión aditiva

Las posibilidades no tienen límites, y están únicamente gobernadas por la imaginación desarrollada por el programador. También aquí la ejecución de una idea, su transformación en un programa, y la aplicación subsecuente es dependiente de una planificación pormenorizada. Es más importante asegurarse que la rutina de tratamiento está efectuando el trabajo que pretendemos haga.

Crea la rutina de tratamiento como un bloque o párrafos preparados, la INposición de los datos como una rutina separada y la presentación de los resultados elaborados también como una rutina separada. Para hacer que la tarea sea más fácil usa nombres de variables que guarden relación con la función que desempeñan.

La rutina del ejemplo, que calculaba el salario anual después de 'impuestos' y de deducciones de la seguridad social está confeccionadas en la Fig. 16.1 a 16.3 como un sistema total explotable. Eso pudiera desarrollarse en un programa en que todos los datos están reflejados inicialmente en la memoria, el procesamiento se efectúa a continuación y los resultados se exponen finalmente en la pantalla, lo que sería un enfoque lógico y estructurado.

Capítulo Diecisiete

Ideas para Aplicaciones

Las aplicaciones para los ordenadores tal y como yo las veo es la única manera de aprovechar y desarrollar inicialmente los ordenadores domésticos. Hoy en día mucha gente puede manipular sus computadoras en casa para mostrar figuras de diversos colores, tanto en movimiento como estáticas, y para hacer animación de figuras por toda la pantalla; sin embargo hay muy pocos que realmente apliquen sus habilidades para programar las tareas contables necesarias en el hogar y en una pequeña oficina. Adivino que la razón para eso no es la falta de aplicaciones posibles sino un grado elevado de incertidumbre sobre como empezar. Para comenzar, examina las ideas sugeridas y estudiadas en profundidad en este libro, además de las áreas de mantener un libro de caja y cualquier otro programa general de contabilidad. Comencemos con el aspecto del cálculo. La cuestión preguntada en este momento en los programas profesionales es la de **¿qué pasa si...?.** Una condición o posibilidad es elegida y el resultado posible se evalúa a partir de hechos conocidos y previamente introducidos en el programa por el usuario. Consideremos las propuestas planteadas en la Fig. 17.1.

Si yo gano pts .... ¿tendría suficiente dinero para comprar un nuevo coche deportivo?
Salario = Pts ....
Coste del artículo = Pts ....
Deducciones
Semana 1 Pts....... Semana 2 Pts ....
Semana 3 Pts....... Semana 4 Pts ....
Deducciones mensuales Pts
Gastos anuales S = Semana 1 + 2 + 3 +4 * 12 = ?
M = Mensual * 12 = ?
Gastos totales = S + M
Respuesta = Ganancias - Gastos
Si respuesta > 0 entonces compro artículo

Figura 16.1: Propuesta de un dilema ¿qué pasa si...?

El enorme potencial de las variables múltiples o tablas ringladas o ristradas, puede explotarse en una de dos áreas: planillas cuadriculadas de cálculo, o bases de datos multidimensionales.

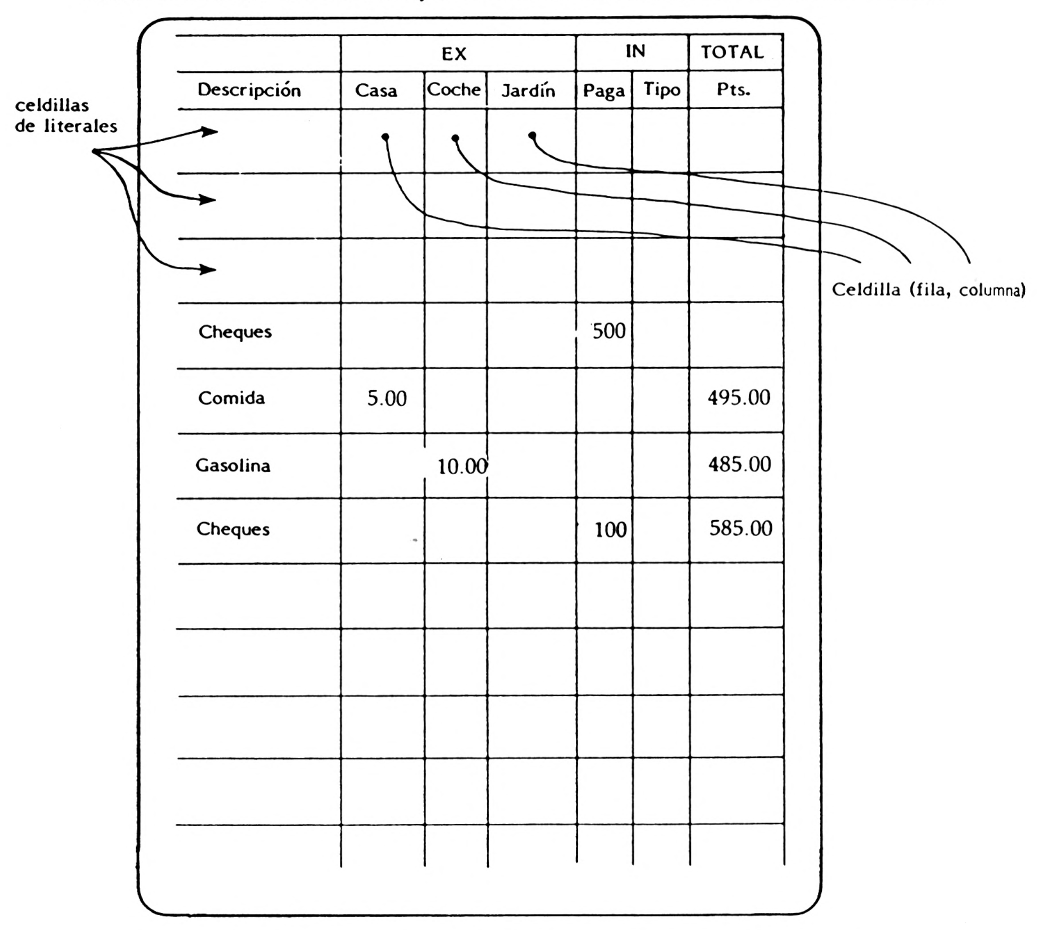


Figura 17.2: Bosquejo de la imagen en pantalla para una planilla de cálculo - propuesta para una aplicación

Las planillas de cálculo

Mediante la utilización de tablas bidimensionales, **ringladas,** una numeral y otro literal compuesta de caracteres cualesquiera, cada una con dos dimensiones, se puede crear una **ringla** rectangular de celdillas o cédulas, cada celdilla vinculada a un área adecuada de la pantalla. La Fig. 17.2 ilustra esta teoría. Las celdillas oblongas que aparecen en la pantalla pueden ser diseñadas por el usuario para que representen datos variables de índole numeral o de índole literal según corresponda.

Los números de **fila** y de **columna** que sirven de referencia para cada celdilla pueden exponerse fácilmente en pantalla mediante un breve programa que sería similar al mostrado en la Fig. 17.3.

```
'El uso del comando EXPONGA ASI... será esencial'

        DIM celda$(3,10), celda (3,10)
          PARA fila igual 0 a 3 de uno en uno
                PARA columna igual 0 a 10, de una en una
                        EXPONGA el valor de celda (fila, columna)
                        Otra ronda de columna
            Otra ronda de fila
            Repita el proceso para aquellas celdas que han sido
        previamente especificadas como para datos literales

        Inscribe los valores para el cálculo
        Realiza el cálculo
        Muestra respuesta y totales
```

Figura 17.3: Posibles métodos de programación

Cada celda puede designarse para contener o bien un dato literal o un bien un dato numeral, pero no ambos. Cada celda numeral puede tener **un valor** inscrito en ella de acuerdo con el tecleado por el usuario como paso previo al proceso de cálculo, o en caso contrario puede que cada celda numérica tenga asignada **una fórmula** matemática cuya evaluación arroje el resultado que ha de inscribirse en ella. La fórmula consistiría de operaciones aritméticas sobre los datos contenidos en otras celdas que han sido designadas como celdas para inscribir directamente los datos tecleados.

Claramente cada área de la pantalla puede asignarse solo para que sea una celda numeral o una celda literal con una sarta de caracteres.

Piensa sobre esta idea y luego trata de desarrollarla en un programa de aplicación que puedas usar posteriormente.

Una base de datos multidimensional

Usando una variable múltiple o tabla ringlada, denominada Person$(10,10) podemos crear una tarjeta cuadriculada con 100 cuadrados marcados en ella. La información reflejada puede corresponder a una persona diferente por cada fila de esa gradilla. Cada columna representaría un concepto o faceta diferente de la información relativa a una persona.

La Fig. 17.4 muestra este enfoque.

NOMBRE	EDAD	CALLE	CIUDAD	PROVINC	SALARIO
GERARDO	29	Toledo, 5	Madrid	Madrid	Pts 90.000

Figura 17.4: Una tarjeta con datos - variables ringladas bidimensionales

Una ampliación posterior pudiera ser la de hacer que la tabla ringlada fuera de tres dimensionales, Person$(100,2,24), que puede considerarse como la información organizada referente a 100 personas distruyéndola en dos páginas para cada persona con 24 campos de información en cada página; o bien podría ser Person$(100, 1, 12, 15) para organizar la información relativa a 100 personas con una página para cada una de ellas, en la que hay 12 áreas de información para cada una de las cuales se reflejarian 12 datos literales diferentes. Ilustrativamente podríamos imaginarlos como en la Fig. 17.6.

Las combinaciones no tienen límite y se puede aplicar a centenares de aplicaciones diferentes. Los cimientos de cada uno de estos programas estarían según las líneas:

1) Preparar la estructura **arreglada** de la base de datos.

2) Definir la índole de cada uno de los campos de información.

3) Inscribir datos en la base de datos, guardarlos en cinta o en disco para el almacenamiento permanente.

4) Usar el programa sobre la base de datos para consultar y tratar la información que se desee.

Manejo de ficheros

Mediante uso de ficheros en cinta o en disco, pueden desarrollarse programas muy interesantes.

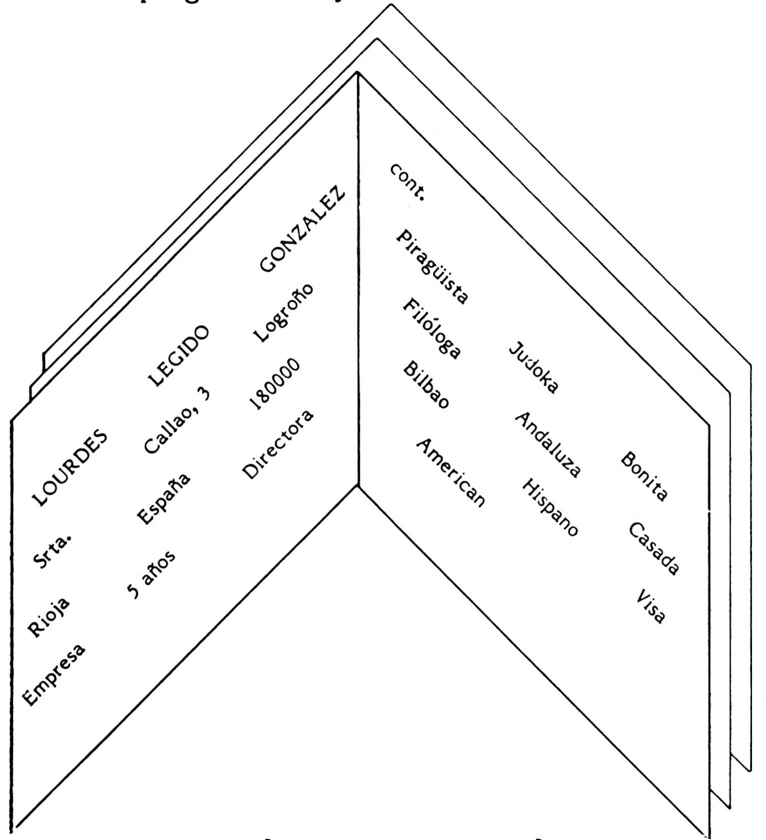


Figura 17.5: Dos páginas de información con 24 campos en cada una

Desafortunadamente un sistema en cinta es mucho más lento y requerirá la manipulación de las cassettes para asegurar que se efectúa la lectura de los datos que se desean de la cinta en el momento oportuno que corresponda según la ejecución del programa. La teoría establece que por ejemplo se prepare una ficha personal con 20 campos de información. Toda esta **ficha** -también llamada **registro**- queda depositada en cinta reseñándola como fichero de datos con un título dado. Se repite luego este proceder para otra ficha, y luego para la siguiente, y así sucesivamente.

Cada vez la información se almacena en la cinta correspondiente. La belleza de todo esto es que la cantidad de fichas que puede almacenarse está limitada solamente por el número de cassettes de cinta que puedes comprar.

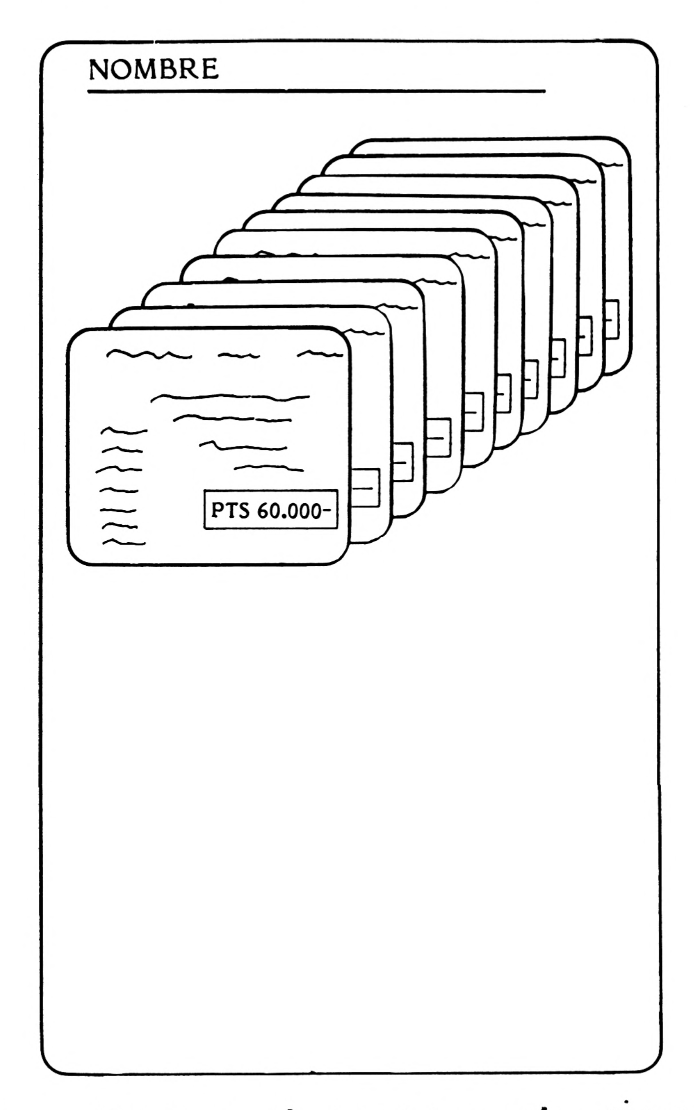


Figura 17.6: Ilustración de la información reflejada en una variable ringlada multidimensional

Ahora hay un pequeño grupo de posibles direcciones a seguir. Piénsalas, estúdialas un poco, y aplícalas a tus necesidades concretas. Empieza con diagramas y bosquejos hasta que estés satisfecho con que el programa que has diseñado satisfará los objetivos que te habías planteado.

Capítulo Dieciocho

Principios Generales y algunos Recordatorios

Al confeccionar programas en BASIC te encontrarás normalmente que hay diversas maneras de obtener el mismo resultado. En la mayoría de los casos no puede decirse cuál es correcta y cuál es equivocada, de manera que usa la que personalmente encuentres más elegante y más comprensible.

Acostúmbrate a usar frecuentemente instrucciones de **memorandum.** Documenta tus programas; sólo te darás cuentas de lo que te quiero decir cuando hayas confeccionado algunos programas, los hayas abandonado durante 2 ó 3 meses y luego intentes estudiarlos para comprender los procedimientos que habías incorporado en cada uno de ellos.

Directrices de programación

Usa la función para inquirir la tecla pulsada, de clave **INKEY$**, dentro de un bucle condicionado **mientras que...,** en lugar de usar instrucciones para pedir e **INponer** datos directamente ahorrándote así la cantidad de veces que suele aparecer en el programa la pulsación de la tecla de **vale**, marcada **ENTER.**

Usa una rutina principal de control desde la cual el curso del programa se desvíe haciendo que **vaya -y venga** al acabar- a la subrutina correspondiente. Usa subrutinas menores para propósitos tan prácticos como el de sacar el mensaje de "pulsa cualquier tecla para continuar".

Enlaza las **apelaciones a** las subrutinas y el comienzo de cada subrutina con una instrucción de memorandum que explique la causa del desvío y la tarea a realizar en él.

Los comandos selectivos de acuerdo **con** un determinado valor de la variable y un **desvío** subsecuente, son muy útiles para la información presentada en forma de menús.

Debieras intentar no usar nunca las instrucciones de **vaya a....** La dirección del curso del programa es muy importante. Los saltos hacia adelante es una acción permisible y puede que a veces ayude a la programación. Los saltos retrospectivos, hacia atrás, pueden llevar a que el programa entre en un 'lazo sin fin' y debieran evitarse dado que en ese caso el control del programa es quitado de las intenciones del programador y del usuario.

La legibilidad de un programa es una de las características más imprescindibles. Las instrucciones de **memorandum** ayudarán al igual que lo hace el uso para las variables de nombres significativos. Es sólo demasiado fácil asignar un nombre con una sola letra a una variable; y de eso también yo soy culpable. Las variables importantes debieran identificarse mediante nombres que guarden estrecha relación con el dato que representan, y en minúsculas normalmente.

'Valida' tus programas de manera que siempre que el usuario tenga una elección pulsando una o más teclas, pulsar cualquiera de las no ofrecidas no tenga ningún efecto sobre el programa. Una manera sencilla de lograrlo es usar: **mientras K$** < > **"B" O K$** < > **"A" repita el bucle.** Así, a no ser que se haya pulsado A o B el bucle condicionado se repetirá ad infinitum.

Habrá de vez en cuando errores del programa que harán que el BASIC finalice la ejecución del programa. Sin embargo hay manera de **resarcirse** de los errores de manera que el programa nunca se la 'pegue'. Eso se hace por la familia de palabras clave del BASIC que está en relación con la **detección de errores.** Los comandos de clave ERROR y de clave ON ERROR harán que el BASIC efectúe una determinada tarea como resultado de haberse cometido un error de la clase especificada o todas las clases posibles de error. Cada clase de error está identificada por un **número** de error especifico. Hay dos funciones de clave ERR y ERL que nos entregan el **número** asignado a esa clase de error y la **línea** del programa donde se detectó el error, respectivamente.

Cuando le mandes que **ocupe** espacio para las variables ringladas de ciertas DIMensiones, sólo reserva para las variables que vas a utilizar ya que sino estarás desperdiciando espacio de memoria.

Al teclear las instrucciones de un programa usa un método para **sangrar**, o 'decalar', 'indentar', las líneas de programa que pertenecen a bucles preconfinados o a bucles condicionados de manera que sean más fácilmente legibles y seas capaz de observar a primera vista donde están las partes repetitivas en el curso del programa.

Finalmente, familiarízate con los instrumentos que tienes a tu disposición. Siempre y cuando te tomes el tiempo necesario para planificar tus esfuerzos y proyectos de programación, lo lograrás. Nunca seas demasiado ambicioso; simplemente recuerda que estás usando un ordenador doméstico de manera que el espacio de memoria puede que finalmente se agote. Pero para eso tendrá que pasar un montón de tiempo.

Apéndice

Apéndice

El Listín Telefónico

Parte uno: El menú de opciones

```
10 REM Programa creador de listin telefonico
20 REM o agregador de nuevos nombres
30 REM y telefonos a un listin
40 REM previamente creado
50 REM Parte segunda   GENERE
60 :
70 LET t=100: LET contador=0: LET fichas=0
80 DIM nome$(t), apel$(t), tfno$(t)
90 LET nbr$="Nombre": LET apl$="Apellidos": LET tfn$="Telefono"
100 MODE 1
110 :
120 GOSUB 210 :REM à Presentar opciones
130 IF K$="A" THEN GOSUB 310 :REM à agregar datos
140 :
150 IF K$="C" THEN GOSUB 620 :REM à crear listin nuevo
160 :
170 GOSUB 960 :REM à guardar listin
180 GOSUB 1190 :REM à explotar listin o PARE
190 END
210 REM Presentador de opciones
220 LOCATE 1,2: PRINT "Elige el utensilio que requieras:"
230 LOCATE 5,4: PRINT "<C>reador de un Nuevo listin"
240 LOCATE 5,6: PRINT "<A>gregador de datos al listin"
250 LOCATE 1,8: PRINT "Pulsa <C> o <A>"
260 WHILE K$<>"A" AND K$<>"C"
270   LET K$=INKEY$
280   LET K$=UPPER$(K$)
290   WEND
300 RETURN
```

Parte dos: Generación del listin

```
310 REM Agregador de datos al listin
320 CLS
330 LOCATE 5,24: PEN 5
340 PRINT "Mete la cinta en la lecto-grabadora"
350 LOCATE 1,1
360 FOR retraso=1 TO 1500: NEXT
370 CLS
380 OPENIN "fonos"
390 WHILE EOF = 0
```

```
400    INPUT#9,nome$(contador)
410    INPUT#9,apel$(contador)
420    INPUT#9,tfno$(contador)
430    PRINT contador,
440    LET contador = contador + 1
450    WEND
460 CLOSEIN
470 LET fichas = contador
480 WINDOW 1,40,1,23: CLS
490 PRINT nbr$,apl$,tfn$
500 LET contador=0
510 WHILE  contador <> fichas
520    PRINT nome$(contador), apel$(contador), tfno$(contador)
530    LET contador = contador + 1
540    WEND
550 WINDOW 1,40,24,24
560 PRINT "Pulsa <S> para que SIGA"
570 WHILE K$ <> "S"
580    LET K$=INKEY$
590    LET K$=UPPER$(K$)
600    WEND
610 LET K$="C": RETURN
620 REM creador de nuevo listin
630 WINDOW 1,40,1,25: CLS
640 PRINT SPC(10): PRINT "INscribe los datos": PRINT
650 PRINT nbr$,apl$,tfn$
660 WHILE contador < 100
670    LET fichas = contador + 1
680    WINDOW 1,40,11,11
690    IF K$=CHR$(32) THEN PEN 2
700    IF K$<>CHR$(32) THEN PEN 5
710    LET K$=""
720    PRINT "Cantidad de fichas: ";fichas
730    WINDOW 1,40,5,9
740    PRINT SPACE$(80)
750    LOCATE 1,2
760    LINE INPUT; nome$(contador)
770    LOCATE 14,2
780    LINE INPUT; apel$(contador)
790    LOCATE 27,2
800    LINE INPUT; tfno$(contador)
810    WINDOW 1,40,17,25
820    PRINT "Pulsa <ENTER> si el dato INscrito VALE"
830    PRINT: PRINT "Si no es correcto pulsa <ESPACIADOR>"
840    PRINT: PRINT "Si quieres que vaya al final pulsa <F>"
850    WHILE K$<>CHR$(13) AND K$<>CHR$(70) AND  K$<>CHR$(32)
860      LET K$=INKEY$: LET K$=UPPER$(K$)
870      WEND
880    CLS
890    IF K$=CHR$(13) GOTO 930
900    IF K$=CHR$(70) GOTO 920
910    IF K$=CHR$(32) GOTO 680
920    LET contador = 99
```

```
930   LET contador = contador + 1
940   WEND
950 RETURN
960 REM guardador del listin creado
970 REM en la lecto-grabadora de cinta
980 PEN 5
990 WINDOW 1,40,1,25: CLS
1000 WINDOW 1,40,11,11
1010 PRINT SPC(5); "Cantidad total de fichas: ";fichas
1020 WINDOW 11,30,20,24
1030 PEN 7: CLS
1040 PRINT SPC(1) "Guardando Listin"
1050 PRINT: PRINT " en la cassette"
1060 WINDOW 1,40,1,8
1070 PEN 1
1080 LOCATE 1,1
1090 OPENOUT "fonos"
1100 LET contador = 0
1110 WHILE contador<fichas
1120   PRINT#9,nome$(contador)
1130   PRINT#9,apel$(contador)
1140   PRINT#9,tfno$(contador)
1150   LET contador = contador + 1
1160   WEND
1170 CLOSEOUT
1180 RETURN
1190 REM Utilizador de listin o PARADOR de curro
1200 WINDOW 1,40,1,25: CLS
1210 PRINT: PRINT "Pulsa la <tecla> apropiada"
1220 LOCATE 5,5: PRINT "<Z>apear := abandonar 'curro'"
1230 LOCATE 5,7: PRINT "<U>tilizar el listin"
1240 WHILE K$<>"Z" AND K$<>"U"
1250   LET K$=INKEY$
1260   LET K$=UPPER$(K$)
1270   WEND
1280 IF K$="Z" THEN CLS
1290 IF K$="U" THEN CLS: RUN "UTILIZ"
1300 RETURN
```

Indice

AMSTRAD CPC 464, 664 Y 6128 PROGRAMACION ESTRUCTURADA.

Para poder dar instrucciones a un ordenador, debemos hacerlo de una forma y con palabras muy precisas, de manera que el micro pueda entenderlas y subsecuentemente ejecutarlas.

Las computadoras Amstrad CPC 6128, 664 y 464 ofrecen varias ventajas para confeccionar programas bien estructurados: nos comunicamos con ellos usando un dialecto BASIC potente y fácilmente comprensible, la documentación en español es excelente, y tanto el 664, como el 6128 están equipados con una unidad ductora de disco para archivar programas y datos en disquette, lo que ahorra mucho tiempo y esfuerzo durante la "traida y llevada" de información.

Este libro lleva al lector a través de cada etapa en el desarrollo de un programa. Y uno de los rasgos sobresalientes del libro es la planificación de ARRIBA hacia ABAJO, i.e: el desglose en tareas del trabajo que ha de efectuar el programa. Se presentan secciones sobre: Introducción al CPC 6128, 664 y 464, La familiaridad inculca la confianza. Los principios del BASIC, Desde pequeños párrafos hasta la programación estructurada y Procesamiento de Textos - que es la clave para el Almacenamiento de Información.

El autor

Stephen Raven es un consultor independiente en Control de Datos. Y anteriormente impartió clases de Computer Studies en el Redbridge College in Essex (Gran Bretaña).

ISBN 84-86381-09-6



CHIQUINQUIRA, 28 (COCUY)
28033 MADRID

AMSTRAD CPC 464, 664 Y 6128
PROGRAMACION ESTRUCTURADA

STEPHEN RAVEN

ra-ma


AMSTRAD
CPC
OCR
MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA
ACME
https://acpc.me/